VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Implementace Shorova r-algoritmu pro nehladkou optimalizaci

On implementation of Shor's r-algorithm
for nonsmooth optimization

Lukáš Kapera

Bakalářská práce

Vedoucí práce: doc. Ing. Petr Beremlijski, Ph.D.

Ostrava, 2021

Poděkování

Děkuji doc. Ing. Petru Beremlijskemu, Ph.D. za vedení při vypracování této bakalářské práce, poskytnutí studijních materiálů a cenné rady. Dále děkuji Ing. Lukáši Pospíšilovi, Ph.D. za inspiraci v oblasti numerické matematiky a moderní výpočetní techniky.

Abstrakt

Nehladkou optimalizací rozumíme nalezení globálních extrémů (minima nebo maxima) funkce, která je sice spojitá, ale v některých bodech svého definičního oboru není diferencovatelná. Tato úloha se často objevuje v mechanice, ekonomii nebo teorii her. Bakalářská práce se zaměřuje na implementaci Shorova r-algoritmu a jeho další optimalizaci tak, aby výsledné řešení bylo spolehlivé, rychlé a přesné.

Práce má následující části:

1. Konvexní analýza,
2. Clarkeův kalkul – zobecnění diferenciálního počtu pro nehladkou funkci,
3. Optimalizace nehladké funkce,
4. Shorův r-algoritmus – představení, implementace a ukázky výpočtu,
5. Numerické experimenty – závislost výpočtu na vstupních datech.

Klíčová slova: Optimalizace**,** minimalizace**,** nehladká funkce, koercivní funkce, Clarkeův kalkul, Lipschitzovská spojitost, zobecněný gradient, subgradient, délka kroku, Shorův r-algoritmus.

Abstract

By nonsmooth optimization we mean finding global extremes (minimum or maximum) of a continuous function, that is not differentiable at some points in its domain. This problem often appears in mechanics, economics and game theory. This Bachelor thesis focuses on implementation of Shor's r-algorithm and its further optimization to achieve reliable, fast and accurate solution.

The Bachelor thesis consists of the following parts:

1. Convex analysis,
2. Clarke's calculus – generalization of the differential calculus for a nonsmooth function,
3. Nonsmooth optimization,
4. Shor's r-algorithm – introducing, implementation and examples of use,
5. Numerical experiments – dependence of the calculation on the input data.

Keywords: Optimization, minimization, nonsmooth function, coercive function, Clarke's calculus, Lipschitz continuity, generalized gradient, subgradient, stepsize, Shor's r-algorithm.

Obsah

Seznam použitých zkratek a symbolů 6
Seznam ilustrací 7
Seznam tabulek 8

1 Úvod 9
2 Základy optimalizace 10
2.1 Klasifikace optimalizačních úloh 12
2.2 Přesnost řešení reálné úlohy 13
2.3 Souhrn poznatků 13
3 Základy konvexní analýzy 14
3.1 Konvexní funkce 19
3.2 Koercivní funkce 21
3.3 Souhrn poznatků 23
4 Úvod do nehladké optimalizace 25
5 Clarkeův kalkul 28
5.1 Lipschitzovská spojitost funkce 28
5.2 Zobecnění diferenciálního počtu 30
5.3 Numerický výpočet zobecněného gradientu 40
5.4 Závěrečné shrnutí 41
6 Minimalizace nediferencovatelné funkce 42
6.1 Nepřímé optimalizační metody 42
6.1.1 Interpolace a aproximace 42
6.1.2 Nepřímá metoda s penalizací 43
6.2 Přímé optimalizační metody 44
6.2.1 Operátor dilatace prostoru – SD 44
6.2.2 Délka kroku daná předem 47
6.2.3 Variabilní délka kroku 48
6.2.4 Ukončovací podmínky výpočtu 50
6.2.5 Metoda největšího spádu – NS 51
6.2.6 Minimalizace s dilatací prostoru – metoda SDG 57
6.2.7 Minimalizace se znalostí funkční hodnoty v minimu 63
6.2.8 Souhrn poznatků a konvergence metod 65

7 Shorův r-algoritmus 66
7.1 Naum Zuselevich Shor 66
7.2 Počítač BESM-6 66
7.3 Základní varianta r-algoritmu 67
7.4 Konvergence r-algoritmu, modifikace r-mikro a r-alfa 74
7.5 Výpočetní náročnost r-algoritmu 75
8 Numerické experimenty 76
8.1 Rosenbrock 77
8.1.1 Výpočetní náročnost a maximální přesnost výpočtu 78
8.1.2 Testování stability výpočtu – 1. část 79
8.1.3 Modifikace r-algoritmu – autodetekce a oprava chybných výpočtů 83
8.1.4 Testování stability výpočtu – 2. část 84
8.1.5 Shrnutí experimentu 86
8.2 Půlměsíc 86
8.2.1 Výpočetní náročnost a maximální přesnost výpočtu 88
8.2.2 Zvýšení spolehlivosti výpočtu - parametr `opakovani` 89
8.2.3 Závislost výpočtu na koeficientu prostorové dilatace α 91
8.3 Charalambous-Bandler 92
8.3.1 Výpočetní náročnost a maximální přesnost výpočtu 93
8.3.2 Závislost charakteristik výpočtu na parametru L 94
8.4 Maxl 97
8.4.1 Závislost průběhu výpočtu na dimenzi úlohy 99
8.5 Neostré globální minimum 101
8.6 Souhrn nejdůležitějších poznatků 104
9 Shrnutí práce 105
10 Použitá literatura 107
11 Seznam příloh 108

Seznam použitých zkratek a symbolů

$[a;\ b]$	Uzavřený interval (odlišně od běžného značení)
$(a; b)$	Otevřený interval (též vektor, zřejmé z kontextu)
$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$	Množina přirozených, celých, racionálních, reálných čísel, $0 \in \mathbb{N}$
$\mathbb{R}^n$, $\Omega \subseteq \mathbb{R}^n$	N-rozměrný euklidovský prostor, množina přípustných řešení úlohy
$\langle a; b \rangle$	Skalární součin vektorů a a b (též lineární obal; zřejmé z kontextu)
$f'(x, v)$	Směrová derivace funkce f (v bodě x ve směru v)
$f^0(x, v)$	Clarkeova směrová derivace funkce f (v bodě x ve směru v)
$\nabla f(x)$	Gradient funkce f (v bodě x)
$\partial f(x)$	Clarkeův zobecněný gradient funkce f (v bodě x) - množina, prvky jsou *Clarkeovy subgradienty*
$\|a\|$	Euklidovská norma vektoru a (lze použít i jiné normy)
■ △	Konec důkazu, konec příkladu
det	Determinant
g, grad	Gradient nebo subgradient (zřejmé z kontextu)
H, J	Hessián, Jacobián
conv{...}	Konvexní obal množiny (obvykle vektorů)
$\|a\|, \|M\|$	Absolutní hodnota čísla a, mohutnost množiny M (počet prvků M)
o	Nulový vektor
D_f, H_f	Definiční obor funkce f, obor hodnot funkce f
$\lceil a \rceil, \lfloor a \rfloor$	Horní (dolní) celá část čísla a
$\bar{x}, \tilde{x}$	Přesné řešení úlohy, přibližné řešení (aproximace přesného)
M^{-1}, M^T	Matice inverzní k matici M, matice transponovaná k matici M

Seznam ilustrací

Obrázek 1 Graf cenové funkce 11
Obrázek 2 Konvexní a nekonvexní množina 14
Obrázek 3 Konvexní polyedr a konvexni polyedrická množina 18
Obrázek 4 Lineární interpolace průhybu nosníku na síti bodů 21
Obrázek 5 Nehladká konvexní cenová funkce 25
Obrázek 6 Minimalizace s využitím derivace zprava 26
Obrázek 7 Minimalizace s využitím derivace zleva 27
Obrázek 8 K příkladu 5.3 29
Obrázek 9 Clarkeův zobecněný gradient hladké funkce 36
Obrázek 10 Clarkeův zobecněný gradient nehladké funkce 36
Obrázek 11 Clarkeův zobecněný gradient pro max funkci 39
Obrázek 12 Rozklad vektoru 45
Obrázek 13 Rutina pro hledání délky kroku 49
Obrázek 14 Srovnání průběhu minimalizace metodou NS pro hladkou a nehladkou funkci 53
Obrázek 15 Graf koercivní funkce 54
Obrázek 16 Průběh minimalizace koercivní funkce 56
Obrázek 17 Naum Z. Shor 66
Obrázek 18 Počítač BESM-6 66
Obrázek 19 Srovnání průběhu výpočtu pro různé koeficienty dilatace prostoru 72
Obrázek 20 Detailní pohled na průběhu výpočtu 73
Obrázek 21 Graf Rosenbrockovy funkce 77
Obrázek 22 Rosenbrock – závislost chyby aproximace na počtu iterací 78
Obrázek 23 Grafu Rosenbrockovy funkce – detailní pohled shora 79
Obrázek 24 Rosenbrock – počáteční aproximace, ve kterých výpočet selhal 80
Obrázek 25 Závislost chyby a počtu iterací na koeficientu prostorové dilatace 82
Obrázek 26 Charakteristiky výpočtu pro modifikovaný r-algoritmus 85
Obrázek 27 Graf funkce Půlměsíc 86
Obrázek 28 Graf funkce Půlměsíc – pohled shora 87
Obrázek 29 Půlměsíc – závislost chyby výpočtu na počtu iterací 88
Obrázek 30 Půlměsíc – závislosti chyby na počáteční aproximaci 89
Obrázek 31 Závislost chyby aproximace na koeficientu prostorové dilatace 91
Obrázek 32 Graf funkce Charalambous-Bandler 92
Obrázek 33 Charalambous-Bandler – závislost chyby výpočtu na počtu iterací 93
Obrázek 34 Graf závislosti spolehlivosti na koeficientu α 95
Obrázek 35 Graf závislosti chyby výpočtu na koeficientu α 95
Obrázek 36 Graf závislosti výpočetní náročnosti na koeficientu α 96
Obrázek 37 Graf funkce Maxl pro dimenzi $d = 2$. 97
Obrázek 38 Maxl – závislost chyby aproximace na počtu iterací 98
Obrázek 39 Závislosti chyby aproximace na dimenzi úlohy – funkce Maxl 99
Obrázek 40 Závislosti výpočetní náročnosti na dimenzi úlohy – funkce Maxl 100
Obrázek 41 Graf funkce s neostrým globálním minimem 101

Seznam tabulek

Tabulka 1 Průhyb nosníku ... 22
Tabulka 2 Řešení příkladu 5.7 pro hladkou funkci ... 31
Tabulky 3 Obecné řešení příkladu 5.7 pro nehladkou funkci ... 32
Tabulka 4 Konkrétní řešení příkladu 5.7 pro nehladkou funkci ... 32
Tabulka 5 K příkladu 5.20 – stanovení množin ... 35
Tabulka 6 K příkladu 5.20 – Clarkeův zobecněný gradient ... 37
Tabulka 7 Kpříkladu 5.20 – Clarkeova směrová derivace ... 38
Tabulka 8 Přehled obvyklého zadání nehladkých cenových funkcí ... 40
Tabulka 9 Nastavení počátečních hodnot pro minimalizaci ... 55
Tabulka 10 Nastavení přesnosti ... 56
Tabulka 11 Srovnání výpočetní náročnosti jednotlivých metod ... 57
Tabulka 12 K příkladu 6.56 – nastavení přesnosti ... 62
Tabulka 13 K příkladu 6.56 – nastavení počátečních hodnot pro minimalizaci ... 62
Tabulka 14 Srovnání výpočetní náročnosti metod ... 62
Tabulka 15 Řešení úlohy 6.56 ... 63
Tabulka 16 Nastavení počátečních hodnot pro minimalizaci ... 64
Tabulka 17 Srovnání výpočetní náročnosti metod při znalosti hodnoty v minimu ... 64
Tabulka 18 Výchozí optimální hodnoty parametrů ... 68
Tabulka 19 Nastavení přesnosti výpočtu ... 69
Tabulka 20 Výpočetní náročnost pro koeficient $\alpha = 2$ ... 69
Tabulka 21 Výpočetní náročnost pro koeficient $\alpha = 2{,}5$ ... 70
Tabulka 22 Výpočetní náročnost pro koeficient $\alpha = 3$ ... 70
Tabulka 23 Výpočetní náročnost pro $\alpha = 3$ s úpravou parametrů p a h. ... 72
Tabulka 24 Zadání Rosenbrockovy funkce ... 77
Tabulka 25 Maximální přesnost minimalizace Rosenbrockovy funkce ... 78
Tabulka 26 Přehled změn pro vstup modifikovaného r-algoritmu ... 83
Tabulka 27 Přehled změn pro výstup modifikovaného r-algoritmu ... 84
Tabulka 28 Hodnoty nově použitých parametrů ... 84
Tabulka 29 Srovnání výsledků původní a modifikované metody ... 85
Tabulka 30 Zadání minimalizace funkce Půlměsíc ... 87
Tabulka 31 Odhad oblasti, ve které leží bod minima ... 88
Tabulka 32 Maximální přesnost minimalizace funkce Půlměsíc ... 88
Tabulka 33 Zadání počtu opakování výpočtu v případě jeho selhání ... 89
Tabulka 34 Výsledky experimentu pro různý počet opakování výpočtu v případě selhání ... 90
Tabulka 35 Zadání Charalambousovy-Bandlerovy funkce ... 92
Tabulka 36 Odhad oblasti, ve které leží bod minima ... 93
Tabulka 37 Maximální přesnost minimalizace Charalambousovy-Bandlerovy funkce ... 93
Tabulka 38 Hodnoty použitých parametrů ... 94
Tabulka 39 Počty chybných výsledků pro jednotlivá měření ... 96
Tabulka 40 Zadání pro funkci Maxl ... 98
Tabulka 41 Zadání Gofflinovy funkce ... 101
Tabulka 42 Zadání a výsledky minimalizace funkce s neostrým globálním minimem ... 103

1 Úvod

V posledních letech zažívá optimalizace velký rozmach. Můžeme se s ní setkat například v mechanice, ekonomice nebo teorii her. Uplatnění nalézá ve výzkumu, plánování i řízení v reálném čase. Úloha optimalizace spočívá v nalezení minima nebo maxima cenové funkce. Nehladkou optimalizací myslíme nalezení minima nebo maxima funkce, která není spojitě diferencovatelná.

Úlohu nehladké optimalizace lze principiálně řešit dvěma způsoby. První, nepřímý, spočívá v převedení úlohy na hladkou optimalizaci, kdy nehladkou cenovou funkci nejdříve aproximujeme hladkou funkcí a posléze úlohu vyřešíme algoritmem pro minimalizaci hladké funkce. Druhý, přímý, spočívá v použití algoritmu přímo vhodného pro minimalizaci nehladkých funkcí. Tyto metody, myšlenkově vycházející z gradientních metod, nazýváme **subgradientní metody**. Představují zobecnění gradientních metod pro nehladké funkce a umožňují nám řešit širší spektrum úloh. Mezi ně spadá i **Shorův r-algoritmus**, který je hlavním tématem této práce.

Pro snazší porozumění tématu nejdříve probereme základní analytické pojmy. V úvodu do nehladké optimalizace probereme **Clarkeův kalkul** a v následujících kapitolách se již věnujeme subgradientním metodám. Postupně implementujeme od jednoduchých až po složitější a sofistikovanější metody, zejména Shorův r-algoritmus. Výhody a nevýhody různých přístupů k řešení minimalizační úlohy předvedeme na praktických úlohách. Nakonec porovnáme náročnost a přesnost jednotlivých metod.

V rámci řešených úloh se zabýváme primárně **korektními úlohami** (mající právě jedno řešení), avšak na jednoduchých příkladech si rovněž ukážeme, jakým způsobem postupovat, pokud kritérium korektnosti splněno není. V rámci ukázkových řešení budeme dodržovat věcné i formální požadavky, mezi které patří především důkaz korektnosti řešení (správnost a konečnost algoritmu). Důraz rovněž klademe na efektivní využití moderních výpočetních systémů.

2 Základy optimalizace

Tato kapitola čerpá zejména z literatury [1] a [3], kterou můžeme vřele doporučit zájemcům o hlubší pochopení probíraného tématu.

Nechť funkce f je zobrazení z podmnožiny Ω konečně rozměrného Euklidovského prostoru $\mathbb{R}^n$ do množiny reálných čísel $\mathbb{R}$

$$f: \Omega \to \mathbb{R}, \Omega \subseteq \mathbb{R}^n. \tag{2.1}$$

Funkci f nazýváme **cenovou funkcí** a obvykle jí získáme při řešení jiné praktické úlohy. Množinu Ω nazýváme **množinou přípustných řešení**. Při porovnání Ω a $\mathbb{R}^n$ mohou nastat tři případy.

$$\begin{cases} \Omega = \mathbb{R}^n, \\ \Omega \neq \mathbb{R}^n \wedge \Omega \neq \emptyset, \\ \Omega = \emptyset. \end{cases}$$

Pokud $\Omega = \mathbb{R}^n$, pak se jedná o úlohu **optimalizace bez omezení**, v opačném případě se jedná o úlohu **omezené optimalizace**. Zvláštní případ nastane, pokud Ω je prázdná množina - pak úloha nemá řešení.

Obecnou **optimalizační úlohou** rozumíme nalezení globálního minima funkce f, což zapisujeme

$$\min_{x \in \Omega} f(x). \tag{2.2}$$

Alternativní úlohou je nalezení takových bodů $x \in \Omega$, kde globální minimum nastává

$$\operatorname{argmin}_{x \in \Omega} f(x). \tag{2.3}$$

V úloze (2.3) hledáme body, ve kterých má cenová funkce globální minimum, tedy body, pro které platí

$$\bar{X} = \operatorname{argmin}_{x \in \Omega} f(x) = \{\bar{x} \in \Omega \mid \forall x \in \Omega: f(x) \geq f(\bar{x})\}. \tag{2.4}$$

Nyní, s pomocí výsledku úlohy (2.4), můžeme zapsat řešení úlohy (2.2)

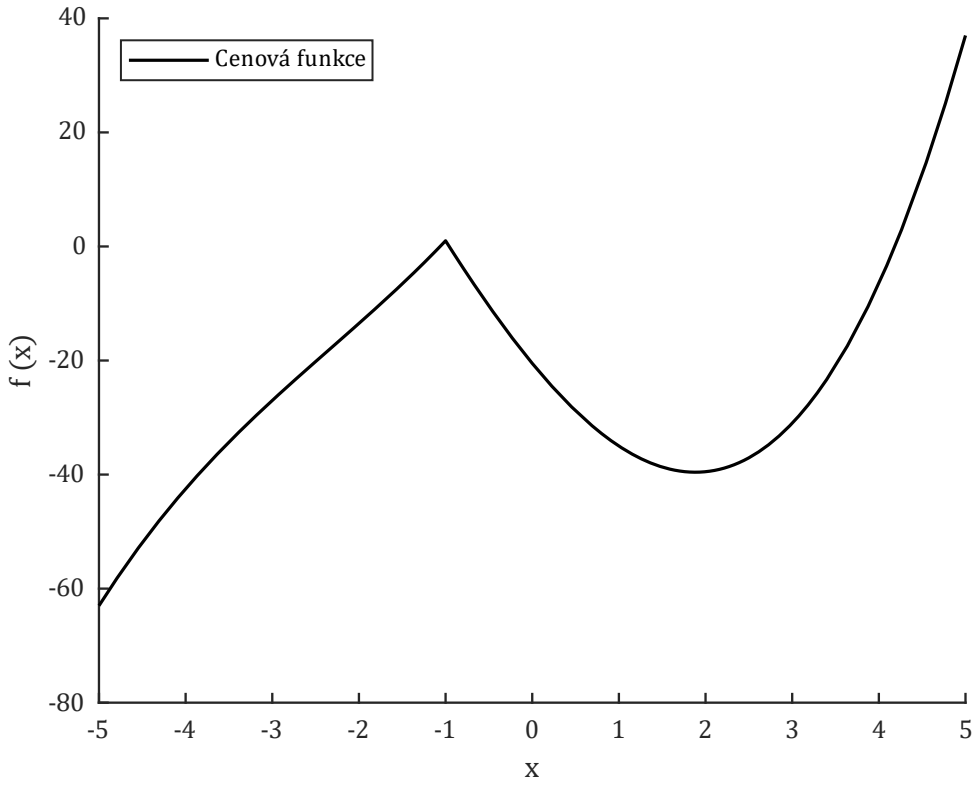
$$G_{min} = \min_{x \in \Omega} f(x) = f(\bar{x}), \text{ kde } \bar{x} \text{ je libovolný prvek } \bar{X}. \tag{2.5}$$

Poznámka: Hledání globálního maxima funkce f je totéž, jako hledání globálního minima funkce $-f$. Pokud máme najít globální maximum, úlohu snadno převedeme na hledání minima – literatura [1].

Mezi úlohami (2.2) a (2.3) se často nerozlišuje. Zjednodušeně řečeno, minimalizací funkce se myslí nalezení jejího minima i bodů, ve kterých funkce tohoto minima nabývá.

Z numerické matematiky víme, že **efektivní algoritmy známe zejména pro korektní úlohy** (úlohy mající právě jedno řešení). To stejné platí i pro úlohu optimalizace. U zcela obecných cenových funkcí je hledání extrémů velmi složité až téměř nemožné, mnohdy jediný funkční algoritmus pro takové funkce představuje systematické prohledávání množiny přípustných řešení (tzv. **triviální optimalizace**). Z tohoto důvodu se vždy snažíme zadanou úlohu převést na množinu korektních dílčích úloh, které vygenerují množinu potenciálních globálních minim, z nichž vybereme to nejmenší. Obecně se jedná o netriviální úlohu. My si ukážeme pouze úlohy, kdy lze rozdělení na dílčí úlohy graficky odhadnout nebo snadno analyticky určit.

Úloha 2.7: Uvažujme cenovou funkci $f: \mathbb{R} \to \mathbb{R}, f = 0{,}5(x-1)^3 + 5x^2 - 20|x+1|$, a dále omezení $x \in [-3, 4]$. Úkolem je najít globální extrémy funkce.



Obrázek 1- Z grafu snadno vyčteme intervaly, ve kterých leží extrémy cenové funkce.

Řešení: Pokud je to možné, vždy nejdříve vykreslíme graf. Z podmínky $x \in [-3, 4]$ určíme množinu přípustných řešení $\Omega = [-3, 4] \neq \mathbb{R}$, úloha tedy vede na omezenou optimalizaci. Protože množina Ω je omezený a uzavřený interval a funkce f je spojitá (byť nehladká), nabývá cenová funkce na množině přípustných řešení Ω globálního minima i maxima (Weierstrassova věta [10]). Extrémy mohou nastat v krajních bodech intervalu $[-3, 4]$ nebo v bodech lokálních extrémů.

Označme P množinu potenciálních globálních extrémů funkce f, dále bod x_{L1}, kde má funkce lokální maximum a bod x_{L2}, kde lokální minimum. Potom $P = \{-3, 4, x_{L1}, x_{L2}\}$.

Z grafu snadno vyčteme, že $x_{L1} \in [-2, 0]$ a $x_{L2} \in [1, 3]$. **To nejpodstatnější ovšem je, že v obou uvedených intervalech leží právě jeden extrém. Tím jsme původní nekorektní úlohu převedli na korektní dílčí úlohy a můžeme použít efektivní algoritmy k jejich řešení.**

Dořešíme úlohu. V bodu x_{L1} má funkce lokální maximum, úlohu tedy převedeme na hledání lokálního minima funkce $g = -f$ na intervalu $[-2, 0]$. Řešením je $\boldsymbol{x_{L1} = -1}$. V bodu x_{L2} má funkce lokální minimum, úlohu nemusíme převádět a řešením je $\boldsymbol{x_{L2} \cong 1,883}$. Získali jsme tak množinu potenciálních řešení $P = \{-3; -1;\ 1{,}883;\ 4\}$. Globální minimum a maximum získáme jako řešení úloh

Úloha	**Řešení**
$G_{min} = \min\{f(p) \mid p \in P\}$	$G_{min} \approx -11{,}91$
$x_{min} = \operatorname{argmin}\{f(p) \mid p \in P\}$	$x_{min} \approx 1{,}883$
$G_{max} = \max\{f(p) \mid p \in P\}$	$G_{max} = 1$
$x_{max} = \operatorname{argmax}\{f(p) \mid p \in P\}$	$x_{max} = -1$

Odpověď: Cenová funkce f má na množině přípustných řešení $\Omega = [-3, 4]$ globální minimum v bodě $x_{min} \cong 1{,}883$, $G_{min} = -11{,}91$ a globální maximum v bodě $x_{max} = -1$, $G_{max} = 1$.

Poznámka: Pokud úlohu neomezíme na interval $[-3, 4]$ a řešíme ji jako optimalizaci bez omezení, pak cenová funkce žádný globální extrém nemá, protože $\lim_{x \to -\infty} f = -\infty$ a $\lim_{x \to +\infty} f = +\infty$. Pokud zvolíme otevřený interval $(-5, 5)$, opět nezískáme žádný globální extrém, protože jednostranné limity $\lim_{x \to -5^+} f = -63$ a $\lim_{n \to +5^-} f = 37$ jsou menší, respektive větší než lokální extrémy, avšak krajní body do množiny přípustných řešení nepatří. A konečně zvolíme-li otevřený interval $(-3, 4)$, řešení úlohy se nezmění, protože oba globální extrémy nastaly v bodech lokálních extrémů, nikoli v krajních bodech intervalu.

△

2.1 Klasifikace optimalizačních úloh

Optimalizovat obecně znamená vyřešit úlohu **matematického programování**. V závislosti na funkci f a množině přípustných řešení Ω, kterou získáme jako řešení soustavy rovnic (2.9) a nerovnic (2.8), rozlišujeme následující programovací úlohy [2].

$$g_k(x) \leq 0, x \in \mathbb{R}^n, k = 1,2, \dots, m, \tag{2.8}$$

$$h_j(x) = 0, x \in \mathbb{R}^n, j = 1,2, \dots, p. \tag{2.9}$$

- **Lineární programování** – pokud jsou všechny funkce f, g_k a h_j lineární,
- **Kvadratické programování** – jedná se o minimalizaci, f je kvadratická, g_k a h_j jsou lineární,
- **Konvexní programování : minimalizace** – f a g_k jsou konvexní a h_j jsou lineární,
- **Konvexní programování : maximalizace** – f je konkávní, g_k jsou konvexní a h_j jsou lineární,
- **Obecné nelineární programování –** v ostatních případech (například u koercivních funkcí).

Konvexní programování může být lineární, kvadratické i obecné nelineární. V následujícím textu se budeme zabývat optimalizací funkcí s konvexní a koercivní množinou přípustných řešení. Zavedeme proto ještě následující třídění minimalizačních úloh.

Z hlediska postupu rozlišujeme minimalizaci

- **Nehladké konvexní funkce,**
- **Nehladké koercivní funkce.**

2.2 Přesnost řešení reálné úlohy

Při výpočtech na počítači aproximujeme čísla iracionální (s nekonečným neperiodickým desetinným rozvojem) čísly s konečným počtem desetinných míst, čímž se dopouštíme jisté chyby, která se nejvíce projeví ve vyhodnocování exaktních rovností. Při práci s reálnými čísly tuto skutečnost musíme brát v potaz. Více o této problematice lze najít v literatuře [6].

Další chyby vznikají nahrazením reálné úlohy matematickým modelem, a tohoto numerickým modelem, kdy často dochází k zanedbání veličin s minimálním vlivem na výsledek. Tyto chyby můžeme snížit vhodnou volbou numerické metody, nicméně zcela odstranit je nelze. **Problém rovněž představuje šíření chyb při výpočtu.** Zejména pak tam, kde provádíme vysoký počet iterací (obecně jakýchkoli matematických operací s aproximovanou veličinou). Z tohoto důvodu **je vhodné volit metody s nižším počtem iterací a operací.**

2.3 Souhrn poznatků

- **Optimalizovat** znamená vyřešit **minimalizační úlohu** na **množině přípustných řešení** zadané soustavou rovnic a nerovnic,
- Rozlišujeme **lineární, kvadratické, konvexní a obecné nelineární programování,**
- V rámci obecného programování budeme uvažovat především **nehladkou koercivní funkci,**
- Efektivní algoritmy existují zejména pro **korektní úlohy**, tedy úlohy s právě jedním řešením,
- Nekorektní úlohu obvykle nejdříve převedeme na dílčí korektní úlohy,
- Hledat maximum funkce f je totéž, jako hledat minimum funkce $-f$,
- Při numerickém řešení úloh musíme brát v potaz vliv chyb a snažit se jej omezit.

3 Základy konvexní analýzy

Nejdříve nadefinujeme základní pojmy a úmluvy. Dříve jsme se zmínili, že úlohy budeme řešit nad konečně rozměrným Euklidovským prostorem. Normou budeme myslet Euklidovskou normu. Celou problematiku prostorů lze najít v [7], zde jen zmiňme, že uvažujeme normovaný lineární (vektorový) prostor konečné dimenze.

Věta 3.1: Kompaktní množina

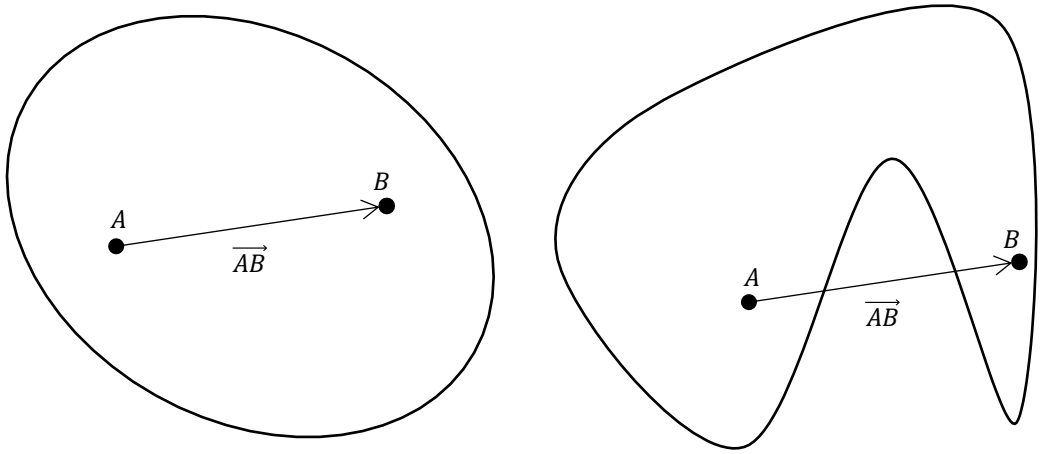
V Euklidovském prostoru konečné dimenze je množina kompaktní právě tehdy, když je uzavřená a omezená (ohraničená).

Důkaz věty vyžaduje zavedení řady dalších pojmů, proto jej zde neuvádíme (najdeme jej v [7]).

Definice 3.2: Konvexní množina

Podmnožina Ω množiny $\mathbb{R}^n$ se nazývá konvexní, jestliže pro každé x a y v Ω a $\alpha \in (0,1)$ leží vektor $s = \alpha x + (1-\alpha)y$ také v Ω.

Prázdná množina je rovněž konvexní. Neformálně řečeno, množina je konvexní, pokud s každou dvojicí bodů obsahuje rovněž úsečku tyto body spojující. Což lze jednoduchými úpravami získat i z definice.



Obrázek 2 – a) vlevo ukázka konvexní množiny, b) množina vpravo konvexní není – neobsahuje celou úsečku AB.

Nechť body $A, B \in \mathbb{R}^n$ a vektor $s \in \mathbb{R}^n$, pak vektor s můžeme zapsat následovně (*Obrázek 2*)

$$\boldsymbol{s} = \alpha B + (1-\alpha)A = \alpha B + A - \alpha A = A + \alpha(B - A) = \boldsymbol{A} + \boldsymbol{\alpha}\overrightarrow{\boldsymbol{AB}}\,. \tag{3.3}$$

Body A a B jsou krajní body úsečky AB, množina $\alpha\overrightarrow{AB}$ představuje její vnitřek. Vztah (3.3) je názornější při vedení důkazů, zdali je množina Ω konvexní.

Mezi **základní konvexní množiny** patří

- Celý prostor libovolné konečné dimenze $\mathbb{R}^n$,
- V prostoru $\mathbb{R}$: Přímka (představuje celý prostor $\mathbb{R}$) a polopřímka,
- V prostoru $\mathbb{R}^2$: Rovina (představuje celý prostor $\mathbb{R}^2$), polorovina a kruh,
- V prostoru $\mathbb{R}^3$: Celý prostor, poloprostor, koule, kužel,
- Analogicky postupujeme u prostorů vyšší dimenze.

Definice 3.4: Uzavřený poloprostor

Uzavřený poloprostor je množina $\{x \in \mathbb{R}^n \mid a^T x \geq c\}$, případně $\{x \in \mathbb{R}^n \mid a^T x \leq c\}$, pro vhodná $a \in \mathbb{R}^n, a \neq \sigma, c \in \mathbb{R}$.

Definice 3.5: Nadrovina

Nadrovina je množina $\{x \in \mathbb{R}^n \mid a^T x = c\}$, pro vhodná $a \in \mathbb{R}^n, a \neq \sigma, c \in \mathbb{R}$.

Uzavřený poloprostor je zobecněním pojmu polopřímky, poloroviny, … pro prostor $\mathbb{R}^n$. V případě ostré nerovnosti získáme (otevřený) poloprostor. Nadrovina je zobecněním pojmu bod, přímka, rovina…. Výraz $a^T x = c$ můžeme psát ve tvaru $a_1x_1 + a_2x_2 + \cdots + a_nx_n = (a_1, a_2, \ldots, a_n) \cdot (x_1, x_2, \ldots, x_n) = \langle a, x\rangle = c$, kde $\langle a, x\rangle$ je skalární součin. Více viz [11].

Příklad 3.6: Nadrovina je zadaná rovnicí $z = 3x + 4y - 10$, určete vhodné parametry a a c.

Řešení: Rovnici upravíme na tvar $3x + 4y - z = 10$, ze kterého vyčteme $a = (3, 4, -1)$ a $c = 10$.

Příklad 3.7: Dokažte, že uzavřený poloprostor $\Omega \subset \mathbb{R}^n$ je konvexní množina.

Řešení: Jedná se o jeden z klíčových důkazů konvexní analýzy. Zvolíme nerovnost $a^T x \geq c$ a podle definice (3.4) pro množinu Ω platí

$$\Omega = \{x \in \mathbb{R}^n \mid a^T x \geq c\}, a, c \in \mathbb{R}, a \neq \sigma. \tag{3.8}$$

Chceme dokázat

$$\forall x_1, x_2 \in \Omega, \forall \alpha \in (0,1): a^T x_1 \geq c \ \wedge \ a^T x_2 \geq c \implies a^T(x_1 + \alpha(x_2 - x_1)) \geq c. \tag{3.9}$$

Levou stranu závěru implikace (3.9) upravíme

$$a^T x_1 + \alpha a^T x_2 - \alpha a^T x_1 = \boldsymbol{a^T x_1 + \alpha(a^T x_2 - a^T x_1)} \tag{3.10}$$

Uvažujme nejdříve variantu

$$\boldsymbol{a^T x_2 \geq a^T x_1 \geq c}. \tag{3.11}$$

Potom

$$a^T x_2 \geq a^T x_1 \implies a^T x_2 - a^T x_1 \geq 0 \implies \boldsymbol{\alpha\left(a^T x_2 - a^T x_1\right) \geq 0}. \tag{3.12}$$

A protože současně platí

$$a^T x_1 \geq c,$$

pak po přičtení výrazu $a^T x_1$ k obou stranám nerovnosti (3.12) získáme

$$a^T x_1 + \alpha(a^T x_2 - a^T x_1) \geq a^T x_1 \geq \mathrm{c} \implies \boldsymbol{a^T x_1 + \alpha\left(a^T x_2 - a^T x_1\right) \geq \mathbf{c}},$$

což jsme chtěli dokázat. Nyní přistupme k druhé možnosti.

Dále uvažujme variantu

$$\boldsymbol{a^T x_1 > a^T x_2 \geq c}. \tag{3.13}$$

Dokažme závěr implikace (3.9), který postupně upravujme

$$a^T x_1 + \alpha(a^T x_2 - a^T x_1) \geq c \mid -a^T x_1,$$

$$\alpha(a^T x_2 - a^T x_1) \geq c - a^T x_1 \ \mid : \alpha > 0,$$

$$a^T x_2 - a^T x_1 \geq \frac{c - a^T x_1}{\alpha}. \tag{3.14}$$

Protože podle předpokladu (3.13) je $a^T x_2 \geq c$, můžeme ve vztahu (3.14) výraz $a^T x_2$ odhadnout zdola výrazem c. Myšlenku vyjádříme matematickým zápisem

$$a^T x_2 \geq c \implies a^T x_2 - a^T x_1 \geq c - a^T x_1,$$

a získáme nerovnost (3.15), kde podle (3.13) platí $a^T x_1 > c \implies c - a^T x_1 < 0$,

$$a^T x_2 - a^T x_1 \geq c - a^T x_1 \geq \frac{c - a^T x_1}{\alpha}. \tag{3.15}$$

A tedy vztahy (3.14) a následně i (3.9) jsou dokázány. Pro $a^T x \leq c$ v definici (3.4) je důkaz analogický.

■

Věta 3.16: Nechť I je neprázdná indexová množina a $X_k \subset \mathbb{R}^n$ jsou konvexní množiny pro všechna $k \in I$. Pak také jejich průnik $\bigcap_{k \in I} X_k$ je konvexní množina.

Důkaz: Důkaz je částečně veden podle [2]. Vycházejme z definice (3.2). Uvažujme body x, y, patřící do průniku všech konvexních množin a $\alpha \in (0,1)$

$$x, y \in \bigcap_{k \in I} X_k \,, \alpha \in (0,1).$$

To však znamená, že tyto body patří do všech jednotlivých konvexních množin X_k a platí

$$\forall k \in I: \alpha x + (1 - \alpha) y \in X_k.$$

Protože dle předpokladu body x, y patří do průniku všech X_k, z uvedeného přímo vyplývá

$$\boldsymbol{\alpha x + (1 - \alpha) y \in \bigcap_{k \in I} X_k}.$$

Což jsme chtěli dokázat.

■

Jelikož uzavřený poloprostor je konvexní množina a průnik konvexních množin je konvexní množina, snadno odvodíme, že **průnik libovolného konečného počtu uzavřených poloprostorů je konvexní množina**. Tato skutečnost má zásadní význam. Uveďme ještě následující definice.

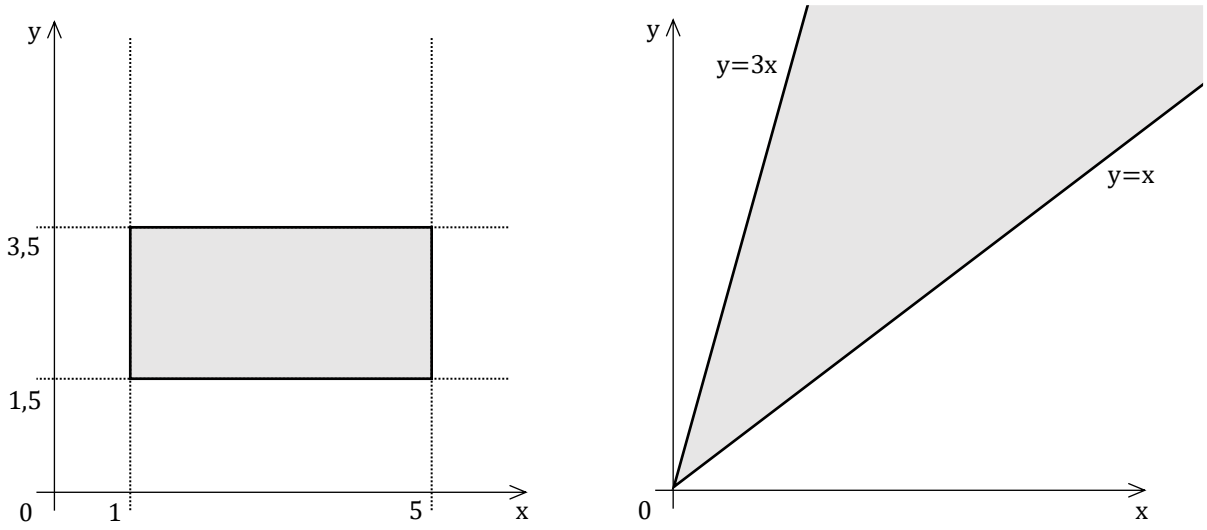
Definice 3.17: Konvexní polyedrická množina

je množina, která vznikne průnikem konečného počtu uzavřených poloprostorů.

Definice 3.18: Konvexní polyedr

Je omezená (uzavřená) konvexní polyedrická množina.

Konvexními polyedry jsou například uzavřený ohraničený interval v $\mathbb{R}$, čtverec v $\mathbb{R}^2$, krychle, kvádr, a jehlan v $\mathbb{R}^3$. Rozdíl mezi konvexním polyedrem a konvexní polyedrickou množinou spočívá v tom, že druhá jmenovaná nemusí být omezená (například *nekonečný* jehlan – intuitivně vnitřek jehlanu včetně pláště, avšak neomezený jeho podstavou). Srovnání vidíme na obrázku 3.



Obrázek 3 – a) vlevo konvexní polyedr, b) vpravo konvexní polyedrická množina.

Příklad 3.19: Určete a charakterizujte množiny na *Obrázku 3*.

Řešení: Ad a) jedná se o konvexní polyedr, protože jde o omezenou uzavřenou množinu, která vznikla jako průnik čtyř polorovin (obecněji poloprostorů)

$$\Omega_A = \{(x, y) \in \mathbb{R}^2 : y \leq 3{,}5 \wedge y \geq 1{,}5 \wedge x \leq 5 \wedge x \geq 1\}.$$

Ad b) jedná se o konvexní polyedrickou množinu, protože jde o průnik dvou polorovin

$$\Omega_B = \{(x, y) \in \mathbb{R}^2 : y \leq 3x \wedge y \geq x\}.$$

Avšak množina Ω_B není omezená, tudíž se nejedná o konvexní polyedr.

△

Poznámka na závěr

Nechť $\Omega \subset \mathbb{R}^n$ je konvexní množina. Pak **uzávěr** i **vnitřek** množiny Ω jsou konvexní množiny. Oproti tomu **hranice** množiny Ω obecně není konvexní množina. Například hranice kruhu je kružnice, která není konvexní.

3.1 Konvexní funkce

Tato kapitola se zaměřuje na konvexní funkce, které mají při řešení úloh řadu příjemných vlastností.

Definice 3.20: Konvexní funkce

Nechť $\Omega \subseteq \mathbb{R}^n$ je konvexní množina. Zobrazení $f: \Omega \to \mathbb{R}$ se nazývá konvexní funkce, je-li její nadgraf konvexní množina, tedy platí-li pro $\forall x, y \in \Omega$ a $\alpha \in (0,1)$:

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y).$$

Definice 3.21: Ryze konvexní funkce

Nechť $\Omega \subseteq \mathbb{R}^n$ je konvexní množina. Zobrazení $f: \Omega \to \mathbb{R}$ se nazývá ryze konvexní funkce, platí-li pro $\forall x, y \in \Omega, \mathrm{x} \neq \mathrm{y}$ a $\alpha \in (0,1)$ ostrá nerovnost

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha) f(y).$$

Definice 3.22: Konkávní funkce

Nechť $\Omega \subseteq \mathbb{R}^n$ je konvexní množina. Zobrazení $f: \Omega \to \mathbb{R}$ se nazývá konkávní funkce, je-li její podgraf konvexní množina, tedy platí-li pro $\forall x, y \in \Omega$ a $\alpha \in (0,1)$:

$$f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha) f(y).$$

Definice 3.23: Ryze konkávní funkce

Nechť $\Omega \subseteq \mathbb{R}^n$ je konvexní množina. Zobrazení $f: \Omega \to \mathbb{R}$ se nazývá ryze konkávní funkce, platí-li pro $\forall x, y \in \Omega, \mathrm{x} \neq \mathrm{y}$ a $\alpha \in (0,1)$ ostrá nerovnost

$$f(\alpha x + (1-\alpha)y) > \alpha f(x) + (1-\alpha) f(y).$$

Podotkněme, že v definicích (3.20) – (3.23) jsme nekladli na funkci f žádné požadavky, pouze jsme žádali konvexní definiční obor. Nezapomínejme, že je-li funkce f konkávní (ryze konkávní), pak funkce $-f$ je konvexní (ryze konvexní) a opačně. To lze snadno dokázat dosazením do definic. Za f dosadíme $-f$ a celou nerovnost vydělíme -1, přičemž otočíme znaménko nerovnosti. Kritéria konvexnosti jsou dobře známá pro diferencovatelné funkce.

Věta 3.24: Nechť $\Omega \subseteq \mathbb{R}^n$ je otevřená konvexní množina a funkce $f: \Omega \to \mathbb{R}$ má spojité druhé parciální derivace v každém bodě Ω. Funkce $\boldsymbol{f}$ **je konvexní** právě tehdy, pokud matice druhých derivací funkce f, *Hessián*, značíme $\nabla^2 f(x)$, je pozitivně semidefinitní v každém bodě $x \in \Omega$. Funkce $\boldsymbol{f}$ **je ryze konvexní** právě tehdy, pokud *Hessián* je pozitivně definitní v každém bodě $x \in \Omega$.

Příklad 3.25: Určete oblasti konvexity funkce $f\colon \mathbb{R}^5 \to \mathbb{R}$, $f(\boldsymbol{x}) = 2x_1^3 + 2x_1x_2 + 2x_2^2 + x_3^2 - 4x_3 + 5x_5^2 + 10$.

Řešení: Jedná se o polynomiální funkci s definičním oborem $D_f = \mathbb{R}^5$, první i druhá derivace existují. Vypočteme první derivace (gradient) a posléze Hessián

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial x_4}, \frac{\partial f}{\partial x_5}\right) = (6x_1^2 + 2x_2, 2x_1 + 4x_2, 2x_3 - 4, 0, 10x_5),$$

$$H = \begin{pmatrix} 12x_1 & 2 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{pmatrix}.$$

Matice je pozitivně semidefinitní právě tehdy, pokud determinanty všech hlavních podmatic $M_i, i = 1,2, \dots ,5$ jsou nezáporné.

$$\det(M_1) = 12x_1, \quad (1)$$
$$\det(M_2) = 48x_1 - 4\,, \quad (2)$$
$$\det(M_3) = 2(48x_1 - 4), \quad (3)$$
$$\det(M_4) = \det(M_5) = 0. \quad (4)$$

Z (2) a (3) dostáváme stejné podmínky $x_1 \geq 1/12$, z (1) $x_1 \geq 0$. Proto **funkce $\boldsymbol{f}$ je konvexní na oblasti** $(\mathbf{1/12}\,;\, \infty) \times \mathbb{R}^{\mathbf{4}}$ (pro ostatní proměnné jsme omezující podmínky nezjistili). Pokud by nás zajímaly **oblasti ryzí konvexity funkce $\boldsymbol{f}$,** tak takové **neexistují**. Hessián této funkce nemůže být v žádném bodu jejího definičního oboru pozitivně definitní – plyne přímo ze (4).

△

Příklad 3.24: Vyšetřete definitnost lineární a kvadratické funkce $f\colon \mathbb{R}^n \to \mathbb{R}$.

Řešení: Lineární funkci, její gradient a Hessián můžeme zapsat ve tvaru

$$f = a_0 + a_1x_1 + \cdots + a_nx_n\,,$$

$$\operatorname{grad}(f) = (a_1, a_2, \dots, a_n),$$

$$H = \mathbf{0}.$$

Hessián lineární funkce je nulová matice, z čehož vyplývá, že lineární funkce je konvexní i konkávní současně. Avšak nikdy nemůže být ryze konvexní ani ryze konkávní.

Hessián obecné reálné kvadratické funkce je nenulová reálná čtvercová matice. U kvadratické funkce tedy můžeme určit definitnost bez ohledu na proměnné. Jinak řečeno, kvadratická funkce je pozitivně definitní, negativně definitní nebo indefinitní vždy na celé množině $\Omega \subseteq \mathbb{R}^n$.

△

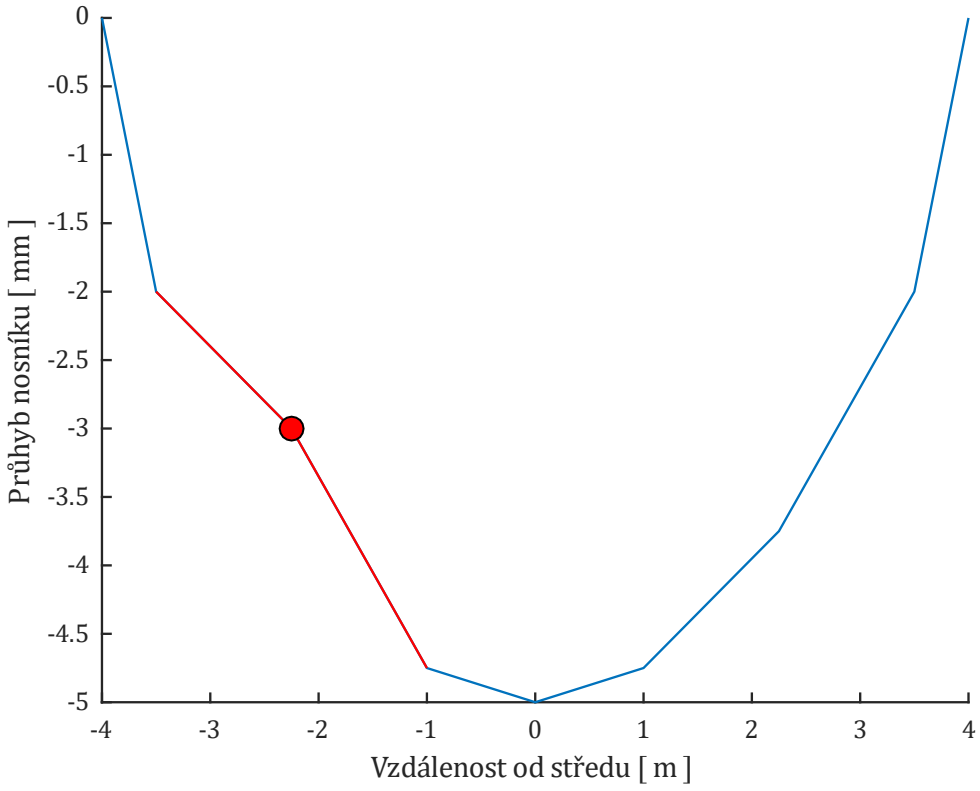
Výhodou konvexních funkcí je, že v každém bodu lokálního minima je současně i globální minimum. V případě ryze konvexní funkce se dokonce jedná o jediné globální minimum (ve smyslu v kolika bodech toto minimum nastane).

Věta 3.27: Nechť $\Omega \subseteq \mathbb{R}^n$ je konvexní množina a $f: \Omega \to \mathbb{R}$ je konvexní funkce. Pak každé lokální minimum funkce f na Ω je současně globálním f na Ω.

Důkaz věty (3.27) najdeme například v [2]. Avšak pozor, věta (3.27) *pouze* říká, že existuje-li lokální minimum, pak se současně jedná o globální minimum (negarantuje jeho existenci).

Nyní se ještě zastavme nad konvexní množinou $\Omega \subseteq \mathbb{R}$. Ta může být otevřená, částečně uzavřená nebo uzavřená. To je při minimalizaci dosti podstatná skutečnost. Pokud se jedná o otevřenou množinu, je situace jednoduchá – hledáme pouze lokální extrémy na vnitřku Ω a posléze vybereme globální minimum. Avšak v případě, kdy Ω obsahuje alespoň část své hranice, je nutné najít i globální minimum této (částečné) hranice a porovnávat jej s lokálními extrémy. Teprve pak můžeme určit globální minimum.

3.2 Koercivní funkce



Obrázek 4 – Lineární interpolace průhybu nosníku na síti bodů.

Začněme motivačním příkladem. Změřili jsme průhyb nosníku (zatíženého uprostřed závažím). Zvolili jsme metodu sítí a získali následující data, která jsme poté lineárně interpolovali.

Číslo měření	1	2	3	4	5	6	7	8	9
Vzdálenost [m]	-4,00	-3,50	-2,25	-1,00	0	1,00	2,25	3,50	4,00
Průhyb [mm]	0	-2,00	-3,00	-4,75	-5,00	-4,75	-3,75	-2,00	0

Tabulka 1– Průhyb nosníku.

Zajímá nás maximální hodnota průhybu, tedy minimum y-ové souřadnice. Jde o minimalizaci nehladké funkce, je tu ovšem jeden problém – minimalizovaná funkce není konvexní, důvodem je měření číslo 3. Pozor, nemusí se nutně jednat o chybu měření, svou roli mohl sehrát nosník.

Představme si, že bychom funkci průhybu nosníku na jeho obou koncích protáhli do nekonečna tak, aby platilo $\lim_{|x|\to\infty} f(x) = +\infty$. Nyní můžeme považovat průhyb nosníku za koercivní funkci.

Výše popsaná situace není vůbec ojedinělá. Naopak, při minimalizaci úloh vycházejících z praxe nemůžeme předpokládat konvexitu naměřených dat, byť teoretický problém konvexní je. Minimalizací právě takovýchto funkcí se budeme dále zabývat. Dodejme, že předpoklad koercivity funkce při její minimalizaci nemusí být vůbec omezující, pokud funkci umíme vhodně dodefinovat (průhyb nosníku), případně ji upravit tak, aby nedošlo ke změně minima. **Koercivní funkce může být hladká i nehladká**. Zásadní podmínka ale zůstává – úloha musí být korektní, tedy mít právě jedno řešení (alespoň na dané konvexní oblasti). Nyní formálněji.

Definice 3.28: Koercivní funkce

Nechť $\Omega \subseteq \mathbb{R}^n$ je neomezená množina a $J\colon \Omega \to \mathbb{R}$. Řekneme, že funkce J je koercivní, jestliže

$$\lim_{\substack{\|\omega\|\to\infty \\ \omega\in\Omega}} J(\omega) = +\infty$$

Věta 3.29: Nechť $\Omega \subseteq \mathbb{R}^n$ je neomezená uzavřená množina a $f\colon \Omega \to \mathbb{R}$ je spojitá koercivní funkce. Pak funkce f nabývá na Ω minima.

Neomezenou uzavřenou množinou máme na mysli množinu, jejíž hranice je po částech buďto uzavřená nebo neomezená (tedy rovna $\pm\infty$), typickým příkladem je polopřímka, polorovina, obecně poloprostor.

Každá konvexní polyedrická množina, která není konvexní polyedr, je neomezená uzavřená množina a podle věty (3.29) na ní koercivní spojitá funkce nabývá minima. Následující věta je přímým důsledkem vět předchozích.

Věta 3.30: Existence řešení

Konvexní funkce na konvexní polyedrické množině nabývá lokálního minima, které je současně globálním minimem. Jeli funkce navíc ryze konvexní, pak je lokálního i globálního minima dosaženo v jediném bodě.

Příklad 3.31: Je každá konvexní funkce $f: \mathbb{R}^n \to \mathbb{R}$ současně koercivní? Je každá ryze konvexní funkce koercivní?

Řešení: Mohlo by se zdát, že koercivita je obecnější (nadřazený) pojem ke konvexitě. Za definiční obor funkce předpokládejme celý n-rozměrný prostor $D_f = \mathbb{R}^n$, tím máme zajištěnu existenci limity

$$\lim_{\substack{\|\omega\|\to\infty \\ \omega\in\mathbb{R}^n}} f(\omega).$$

Ale pozor! Uvedené tvrzení pro konvexní funkci neplatí. Snadno najdeme protipříklad, například konvexní funkci $f(x) = x$. Protože limita

$$\lim_{x\to-\infty} f(x) = \lim_{x\to-\infty} x = -\infty \neq \infty,$$

funkce $f(x) = x$ není koercivní. Tvrzení dokonce neplatí ani pro ryze konvexní funkce, zde uveďme ryze konvexní funkci $g(x) = e^x$, jejíž limita

$$\lim_{x\to-\infty} g(x) = \lim_{x\to-\infty} e^x = 0 \neq \infty.$$

Ani funkce g tedy není koercivní. V obou případech jsem navíc uvedli zcela běžné, často se vyskytující funkce.

△

Poznámka: Pokud řešíme optimalizací s omezením, tedy hledáme řešení na oblasti, která je, byť jen z části, omezená, může se nám stát, že minimalizační metoda během výpočtu z této oblasti *vyskočí.* Tuto možnost je vždy potřeba ošetřit.

3.3 Souhrn poznatků

Ve shrnutí se zabývejme pouze optimalizační úlohou vedoucí na konvexní funkci f a konvexní množinu přípustných řešení Ω. Popíšeme situace, které mohou nastat.

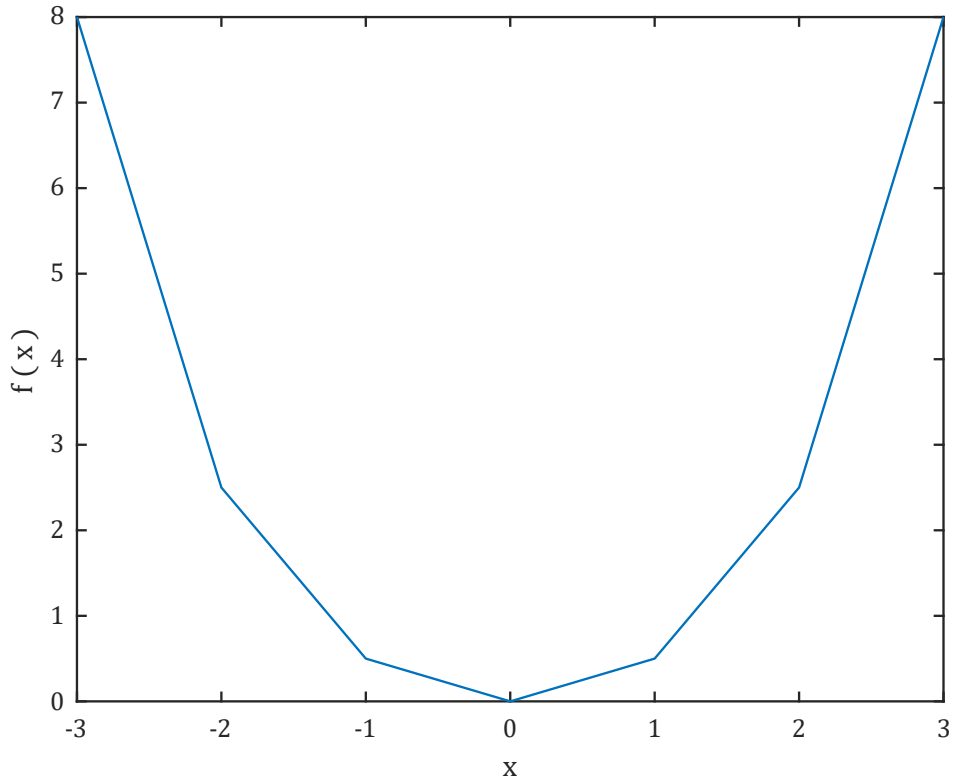
- Je-li Ω uzavřená a omezená, pak řešení existuje vždy:
    - Je-li f ryze konvexní, pak existuje jediné řešení,
    - Není-li f ryze konvexní, pak může existovat více řešení.

- Je-li Ω uzavřená, ale není omezená, pak řešení nemusí existovat, avšak
    - Je-li Ω konvexní polyedrická množina a dále
        - Je-li f koercivní, pak řešení vždy existuje,
            - Je-li f navíc ryze konvexní, řešení existuje jediné.
- Není-li Ω uzavřená, řešení nemusí existovat.
- Je-li Ω prázdná množina, pak řešení neexistuje.

4 Úvod do nehladké optimalizace

V následujících kapitolách se budeme zabývat minimalizací nehladké, avšak spojité cenové funkce. S nehladkou optimalizací se můžeme setkat v různých oborech, například při **řešení úloh z mechaniky, ekonomie nebo teorie her**.

Jiným využitím je **přímý přístup k datům**. Představme si výsledky fyzikálního měření. Chceme určit nejmenší, respektive největší naměřené hodnoty. Pokud použijeme přímý přístup, tedy pracujeme pouze s naměřenými hodnotami, odpadá nutnost aproximace. Naměřené hodnoty stačí lineárně interpolovat, situace je znázorněna na *Obrázku 5*.



Obrázek 5 – Ukázka jednoduché nehladké konvexní cenové funkce.

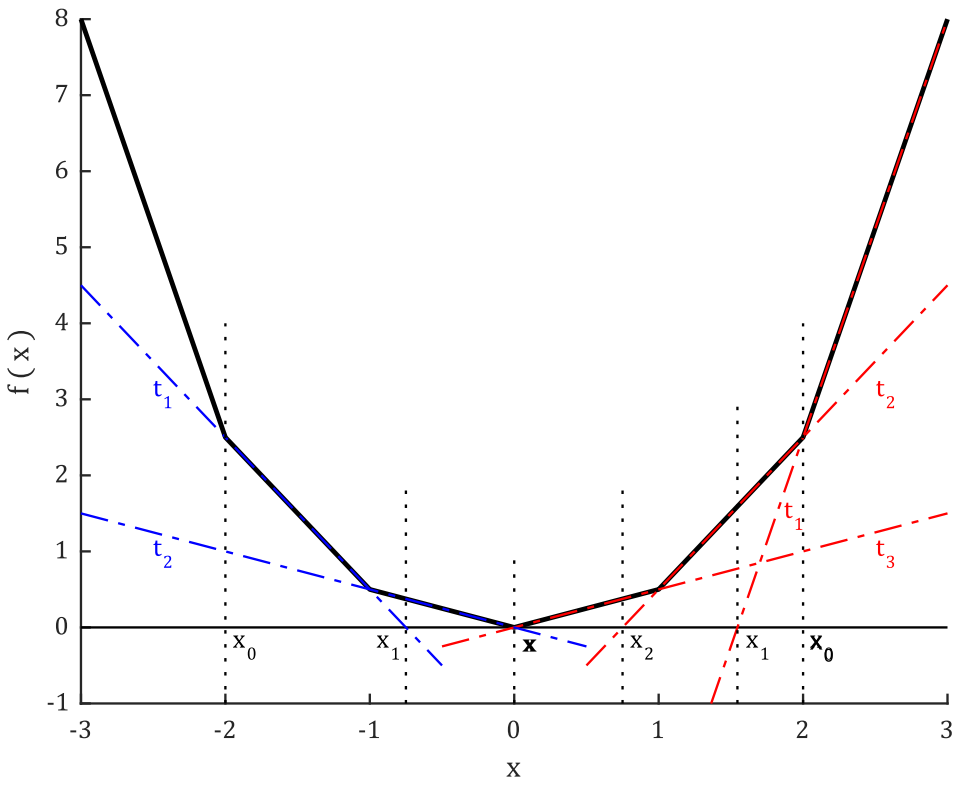
Nehladkou optimalizační úlohu můžeme principiálně řešit dvěma způsoby. **Nepřímá metoda** spočívá v aproximaci cenové funkce hladkou funkcí a následné minimalizaci. Druhou skupinu řešičů představují **přímé metody**, kde k prvotní aproximaci nedochází.

Nabízí se otázka, zdali je možné nějak využít nebo rovnou použít metody optimalizace pro hladkou funkci. Hladké gradientní metody obecně vycházejí z gradientu, tedy z existence prvních derivací, některé vyžadují i derivace vyšších řádů. Problém tak nastává v bodech, kde první (a logicky ani žádná

jiná) derivace neexistují. Dále je tu otázka délky kroku a ukončovací podmínky. Nicméně algoritmus hledání minima pro hladké funkce by v principu použít šel. Částečně nám odpoví následující odstavce.

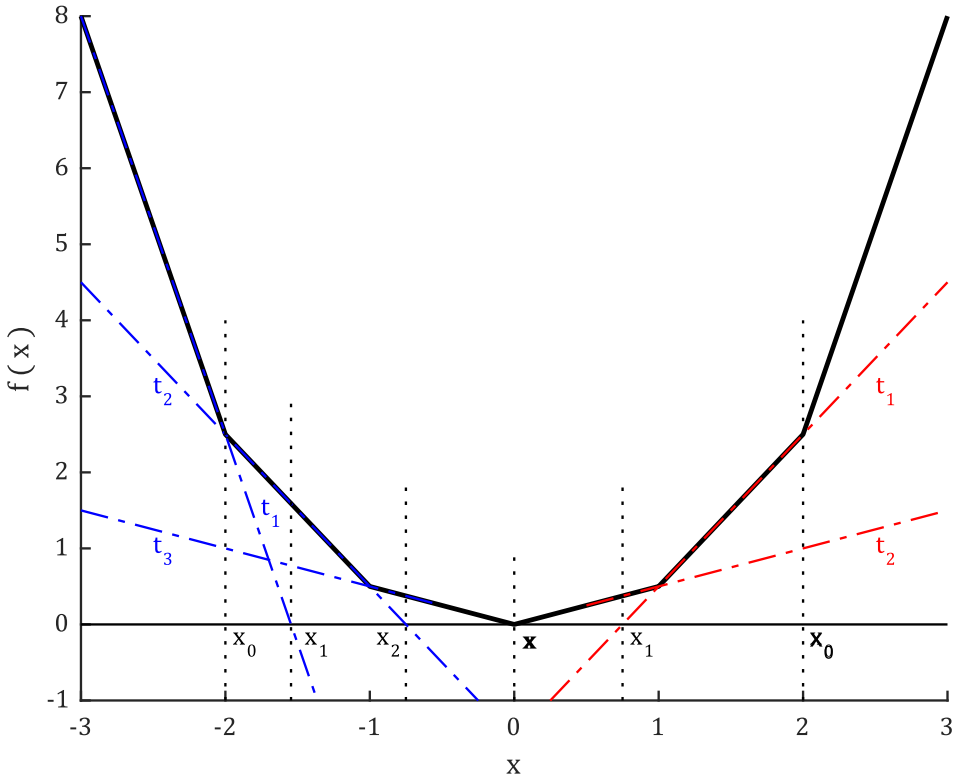
Zajímavým termínem je „*existence numerické derivace*“. S tímto pojmem se můžeme setkat v některých literaturách. Existencí numerické derivace se zde rozumí, že ji počítač dokáže v daném bodě spočítat. To ovšem není příliš šťastná terminologie. Uvědomme si, že derivace v konkrétním bodě buďto existuje nebo neexistuje. Derivace nemůže analyticky neexistovat a současně numericky existovat, to je nesmysl. „*Numericky*“ zde značí pouze způsob výpočtu.

Podívejme se na *Obrázky 6 a 7*. Zvolme počáteční aproximaci $x_0 = -2$. V bodě x_0 určitě existují obě jednostranné vlastní derivace. Při určení směru minimalizace vyjdeme postupně z obou jednostranných derivací. Stejnou úvahu použijeme pro počáteční aproximaci v bodě $x_0 = +2$.



Obrázek 6 – Postup řešení s využitím derivace zprava.

Dořešme motivační úlohu. Zvolíme grafické řešení. Pro určení směru poklesu v daném bodě použijeme nejdříve derivaci zprava. Postup minimalizace je zobrazen pro počáteční aproximaci $x_0 = -2$ modře a pro $x_0 = +2$ červeně, *Obrázek 6*. Vidíme, že obě počáteční aproximace vedou ke správnému řešení úlohy $\operatorname{argmin} f(x) = 0$. Použijeme-li derivaci zleva, nákres je pouze osově převrácený podél osy y, jinak je řešení stejné. Situace je znázorněna na *Obrázku 7*.



Obrázek 7 – Postup řešení s využitím derivace zleva.

Zdá se, že pro minimalizaci nehladké funkce můžeme využít obou jednostranných derivací, rozdíl bude *pouze* v rychlosti řešení. Ano, je tomu skutečně tak. Dokonce jsme mohli použít i libovolnou hodnotu ležící mezi oběma derivacemi.

Zároveň jsme objasnili, proč například metoda největšího spádu může fungovat i pro nehladkou spojitou funkci, za podmínky numerického výpočtu derivace. Výpočet totiž obvykle neodhalí, že funkce v bodě není hladká, protože derivaci počítá z jednoho nebo několika okolních bodů. Otázkou zůstává – můžeme uvedený postup použít i u vyšších dimenzí? Odpověď najdeme v následujících kapitolách.

5 Clarkeův kalkul

Než se pustíme do řešení úloh, probereme potřebné teoretické základy. Clarkeův kalkul zobecňuje diferenciální počet pro nehladkou funkci. Seznámíme se s důležitými pojmy jako **zobecněný gradient** a **Lipschitzovsky spojitá funkce**. Definice ilustrujeme na řešených příkladech.

5.1 Lipschitzovská spojitost funkce

Začněme definicí lipschitzovské spojitosti [1].

Definice 5.1: Lipschitzovská spojitost

Nechť množina $M \subseteq \mathbb{R}^n$, funkce $f : \mathbb{R}^n \to \mathbb{R}$ a $K \in \mathbb{R}_0^+$:

i. Nechť pro funkci f je splněna podmínka

$$\forall x_1, x_2 \in M: |f(x_1) - f(x_2)| \leq K\|x_1 - x_2\|$$

Pak řekneme, že funkce je ***lipschitzovsky spojitá*** s modulem K na množině M.

ii. Nechť pro funkci f je splněna podmínka

$$\exists \varepsilon > 0 \ \forall x_1, x_2 \in x + \varepsilon B: : \ |f(x_1) - f(x_2)| \leq K\|x_1 - x_2\|$$

Kde B je jednotková koule. Pak řekneme, že funkce f je ***lipschitzovsky spojitá v okolí bodu*** $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$ s modulem K.

iii. Nechť funkce f je lipschitzovsky spojitá v okolí každého bodu $x \in \mathbb{R}^n$. Pak řekneme, že funkce f je ***lokálně lipschitzovsky spojitá*** na $\mathbb{R}^n$

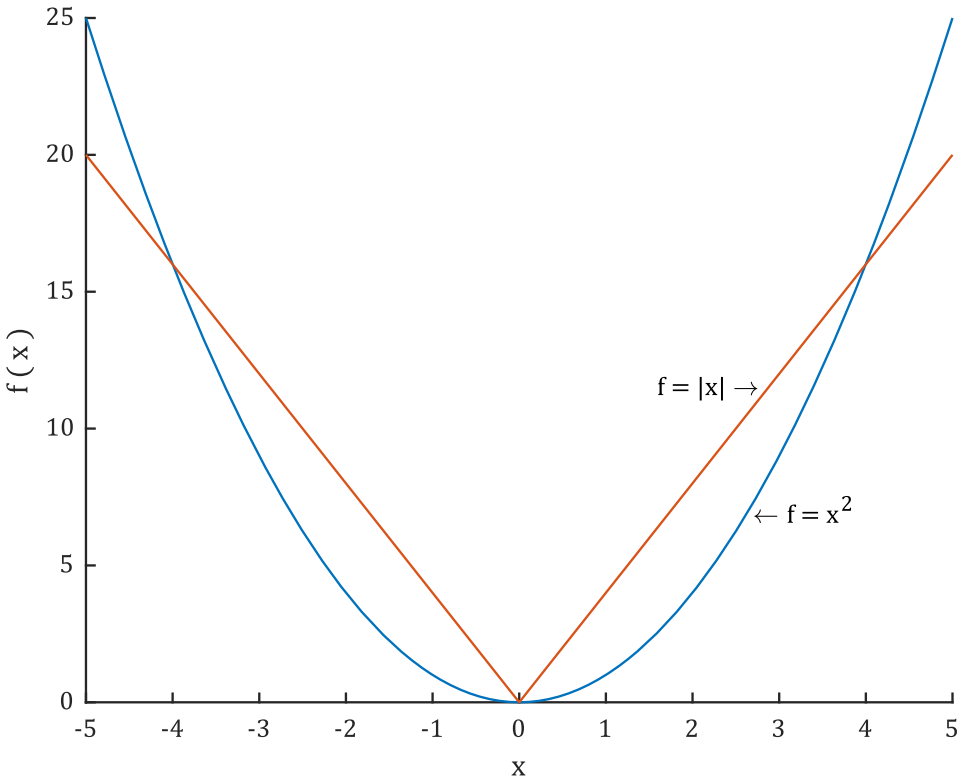
Lipschitzovská spojitost je silnější vlastnost než lokální lipschitzovská spojitost. V čem spočívá rozdíl? Upravme podmínku z definice tak, že osamostatníme K na pravé straně nerovnosti pro $x_1 \neq x_2$

$$\frac{|f(x_1) - f(x_2)|}{\|x_1 - x_2\|} \leq K. \tag{5.2}$$

Výraz na levé straně nerovnice (5.2) není nic jiného, než směrnice přímky spojující body x_1 a x_2. Definice (5.1) požaduje, aby tato směrnice byla vlastní, tedy nesmí se rovnat plus ani minus nekonečnu, a to buď pro libovolné dva body definičního oboru funkce f (lipschitzovská spojitost) nebo pro každé dva body z okolí každého bodu (lokální lipschitzovská spojitost). Protože zkoumáme spojité funkce, stačí vyšetřit chování funkce pouze v hranicích množiny M, respektive existenci asymptot v nekonečnu. Pro naše potřeby obvykle postačí lokální lipschitzovská spojitost. Pro úplnost dodejme, že pro $x_1 = x_2$, a tedy $f(x_1) = f(x_2)$, je podle definice splněna podmínka lokální i celkové lipschitzovské spojitosti.

Příklad 5.3: Vyšetřete lipschitzovskou spojitost funkcí $f_1 = x^2$ a $f_2 = |4x|$ na $\mathbb{R}$.

Řešení: Nejdříve vykreslíme grafy obou funkcí, obě jsou definované na $D_{f1,2} = \mathbb{R} = M$.



Obrázek 8 – Příklad 5.3.

Pro libovolný bod $x_1 \in \mathbb{R}$ a bod $x_2 \to \pm\infty$ vypočteme limitně hodnotu výrazu (5.2) pro f_1

$$\lim_{\substack{x_1 \in \mathbb{R} \\ x_2 \to \pm\infty}} \frac{|x_1^2 - x_2^2|}{\|x_1 - x_2\|} = \lim_{\substack{x_1 \in \mathbb{R} \\ x_2 \to \pm\infty}} \frac{|(x_1 - x_2)(x_1 + x_2)|}{|x_1 - x_2|} = \lim_{\substack{x_1 \in \mathbb{R} \\ x_2 \to \pm\infty}} |x_1 + x_2| = |\pm\infty| = +\infty.$$

Asymptota v nekonečnu neexistuje, funkce nevyhovuje bodu definice (5.1.i), a tudíž není lipschitzovsky spojitá na $\mathbb{R}$. Nyní zvolme $x_1, x_2 \in x + \varepsilon B$. V případě jedné dimenze body náleží do ohraničeného uzavřeného intervalu $x_1, x_2 \in [x - \varepsilon, x + \varepsilon]$ a

$$\lim_{x_1, x_2 \in [x-\varepsilon, x+\varepsilon]} \frac{|x_1^2 - x_2^2|}{\|x_1 - x_2\|} = \lim_{x_1, x_2 \in [x-\varepsilon, x+\varepsilon]} |x_1 + x_2| \leq \lim_{x_1, x_2 \in [x-\varepsilon, x+\varepsilon]} (|x_1| + |x_2|) \leq 2(|x| + \varepsilon).$$

Podstatné je, že $x \in \mathbb{R}$ je konečné číslo, nikoli nekonečno. Pak i $K = 2(|x| + \varepsilon) \in \mathbb{R}$. Jsou tedy splněny body definice (5.1.ii-iii), **funkce $f_1 = x^2$ je lokálně lipschitzovsky spojitá**. Nyní postupujme stejně pro funkci f_2.

$$\lim_{\substack{x_1\in\mathbb{R}\\ x_2\to\pm\infty}} \frac{\big||4x_1|-|4x_2|\big|}{\|x_1-x_2\|} = \lim_{\substack{x_1\in\mathbb{R}\\ x_2\to\pm\infty}} \frac{4\big||x_1|-|x_2|\big|}{|x_1-x_2|} = \lim_{\substack{x_1\in\mathbb{R}\\ x_2\to\pm\infty}} \frac{4\left|\left|\frac{x_1}{x_2}\right|-1\right|}{\left|\left|\frac{x_1}{x_2}\right|-1\right|} = \lim_{\substack{x_1\in\mathbb{R}\\ x_2\to\pm\infty}} \frac{4|0-1|}{|0-1|} = 4 = K.$$

Podmínka (5.1.i) je splněna, **funkce $f_2 = |x|$ je lipschitzovsky spojitá s modulem $K = 4$, z čehož současně plyne i lokální lipschitzovská spojitost.**

△

Věta 5.4: Rademacher

Nechť funkce $f : \mathbb{R}^n \to \mathbb{R}$ je lokálně lipschitzovsky spojitá na $\mathbb{R}^n$ a nechť $D = \{x \in \mathbb{R}^n \mid f$ je diferencovatelná v $x\}$. Potom Lebesguova míra μ množiny bodů, ve kterých f není diferencovatelná, tj. množiny $(\mathbb{R}^n \backslash D)$, je nula.

Rademacherova věta nám říká, že lokálně lipschitzovsky spojité funkce jsou diferencovatelné téměř všude. Právě takové funkce budeme dále studovat.

5.2 Zobecnění diferenciálního počtu

Začneme opět definicemi.

Definice 5.5: Clarkeova zobecněná směrová derivace

Nechť funkce $f : \mathbb{R}^n \to \mathbb{R}$ je lipschitzovsky spojitá v okolí bodu x a nechť vektor $v \in \mathbb{R}^n$. *Clarkeova zobecněná směrová derivace* funkce f v bodě x ve směru v, kterou značíme $f^0(x, v)$, je definována předpisem

$$f^0(x,v) = \limsup_{y\to x, t\downarrow 0} \frac{f(y+tv)-f(y)}{t}$$

Kde y je vektor v $\mathbb{R}^n$ a t je kladné reálné číslo.

Definice 5.6: Clarkeův zobecněný gradient

Nechť funkce $f : \mathbb{R}^n \to \mathbb{R}$ je lipschitzovsky spojitá v okolí bodu x. *Clarkeův zobecněný gradient* funkce f v bodě x, který značíme $\partial f(x)$, je množina

$$\partial f(x) = \{\zeta \in \mathbb{R}^n \mid f^0(x,v) \geq \langle \zeta, v\rangle \ \forall v \in \mathbb{R}^n\}$$

Prvky $\partial f(x)$ nazýváme *Clarkeovými subgradienty* funkce f v bodě x.

Místo *Clarkeova zobecněného gradientu* se také můžeme setkat s názvem *subdiferenciál*. Definice (5.5) a (5.6) lze použít i pro spojitě diferencovatelné funkce, pak $f^0(x,v) = f'(x,v)$ a $\partial f(x) = \{\nabla f(x)\}$.

Příklad 5.7: Určete Clarkeovu směrovou derivaci a Clarkeův zobecněný gradient funkcí $f_1 = x^2$ a $f_2 = |4x|$ v bodech $x_1 = 0, x_2 = 3, x_2 = -3$ v obou směrech.

Řešení: Základní předpoklad, lokální lipschitzovskou spojitost, jsme již dokázali v minulé úloze. Jedná se o jednorozměrnou úlohu, tudíž možné směry jsou dva (jednorozměrné vektory)

$$v_1 = (1) \quad \ominus \to \oplus\,,$$
$$v_2 = (-1) \quad \oplus \to \ominus\,.$$

Nejdříve vypočítáme směrové derivace pro funkci f_1. Výpočet povedeme obecně, poté dosadíme.

$$f_1^0(x,v) = \limsup_{y\to x, t\downarrow 0} \frac{(y+tv)^2 - y^2}{t} = \limsup_{y\to x, t\downarrow 0} \frac{y^2 + 2ytv + t^2v^2 - y^2}{t} = \limsup_{y\to x, t\downarrow 0} \frac{t^2v^2 + 2ytv}{t} =$$

$$\limsup_{y\to x, t\downarrow 0} (tv^2 + 2yv) = 2xv.$$

$\boldsymbol{f_1^0(x,v) = 2xv}$	$v_1 = (1)$	$v_2 = (-1)$
$x_1 = -3$	-6	$+6$
$x_2 = 0$	0	0
$x_3 = +3$	$+6$	-6

Tabulka 2 – Řešení příkladu 5.7- Část pro hladkou funkci.

Nyní spočítejme stejným způsobem směrové derivace pro funkci f_2

$$f_2^0(x,v) = \limsup_{y\to x, t\downarrow 0} \frac{|y+tv| - |y|}{t}\,.$$

Zde narážíme na problém s absolutní hodnotou, obecný výpočet se rozpadá do několika částí. Nejdříve uvažujme $x > 0$, poté $x < 0$.

$$f_2^0(x>0,v) = \limsup_{y\to x, t\downarrow 0} \frac{4|y+tv| - 4|y|}{t} = \limsup_{y\to x, t\downarrow 0} \frac{4y + 4tv - 4y}{t} = \limsup_{y\to x, t\downarrow 0} \frac{4tv}{t} =$$

$$= \limsup_{y\to x, t\downarrow 0} 4v = 4v,$$

$$f_2^0(x<0,v) = \limsup_{y\to x, t\downarrow 0} \frac{4|y+tv| - 4|y|}{t} = \limsup_{y\to x, t\downarrow 0} \frac{-4y - 4tv + 4y}{t} = \limsup_{y\to x, t\downarrow 0} \frac{-4tv}{t} =$$

$$= \limsup_{y \to x, t \downarrow 0} -4v = -4v.$$

Zbývá dořešit možnost $x = 0$, což však nelze současně pro oba vektory v_1 a v_2.

$$f_2^0(0,(1)) = \limsup_{y \to x, t \downarrow 0} \frac{4|y + tv| - 4|y|}{t} = \limsup_{y \to x, t \downarrow 0} \frac{4y + 4tv - 4y}{t} = \limsup_{y \to x, t \downarrow 0} \frac{4tv}{t} =$$

$$= \limsup_{y \to x, t \downarrow 0} 4v = 4v,$$

$$f_2^0(0,(-1)) = \limsup_{y \to x, t \downarrow 0} \frac{4|y + tv| - 4|y|}{t} = \limsup_{y \to x, t \downarrow 0} \frac{-4y - 4tv + 4y}{t} = \limsup_{y \to x, t \downarrow 0} \frac{-4tv}{t} =$$

$$= \limsup_{y \to x, t \downarrow 0} -4v = -4v.$$

Všimněme si, že odstranění absolutní hodnoty výrazu $4|y|$ zde záleží na směru, ve kterém $y \to x = 0$. V případě směru $v = (1)$ jdeme k nule zprava, tedy přes kladná y, proto $4|y| = 4y$. Počítáme-li směr $v = (-1)$, jdeme k nule zleva, tedy přes záporná y, proto $4|y| = -4y$. Na tuto skutečnost musíme brát vždy zřetel, v opačném případě riskujeme nesmyslné výsledky. Nyní shrňme obecné výsledky.

$\boldsymbol{f_2^0(x,v)}$	$v_1 = (1)$	$v_2 = (-1)$
$x < 0$	$-4v$	
$x = 0$	$4v$	$-4v$
$x > 0$	$4v$	

Tabulka 3 – Obecné řešení příkladu 5.7- Část pro nehladkou funkci.

Po dosazení za x_1, x_2 a x_3 získáme hodnoty Clarkeovy zobecněné směrové derivace v těchto bodech.

$\boldsymbol{f_2^0(x,v)}$	$v_1 = (1)$	$v_2 = (-1)$
$x_1 = -3$	-4	$+4$
$x_2 = 0$	$+4$	$+4$
$x_3 = +3$	$+4$	-4

Tabulka 4 – Konkrétní řešení příkladu 5.7- Část pro nehladkou funkci.

Snadno si ověříme, že pro hladkou funkci f_1 platí

$$f_1^0(x,v) = f_1'(x,v).$$

Pro nehladkou funkci f_2 stejný vztah určitě neplatí, můžeme ovšem psát

$$\forall x \in (\mathbb{R}\backslash D)\colon f_2^0(x,v) = f_2'(x,v), D = \{0\}. \tag{5.8}$$

V souladu s větou (5.4) platí, že Lebesguova míra množiny $D = \{0\}$ je v rámci prostoru $\mathbb{R}$ rovná nule. Vztah (5.8) můžeme zobecnit pro libovolnou lokálně lipschitzovsky spojitou funkci f.

Zbývá určit **Clarkeův zobecněný gradient**. Opět vycházejme z jeho definice (5.6). Nebudeme ukazovat výpočet všech gradientů, platí $\partial f_1(x) = \{\nabla f_1(x)\}$ a pro $x \neq 0$ i $\partial f_2(x) = \{\nabla f_2(x)\}$. Ukážeme si pouze výpočet obou zobecněných gradientů právě v bodě $x = 0$. Zjednodušme vztah z definice pro $\mathbb{R}^n = \mathbb{R}^1$.

$$\partial f(0) = \{\zeta \in \mathbb{R} \mid f^0(0,v) \geq \langle \zeta, v\rangle \; \forall v \in \mathbb{R}\}. \tag{5.9}$$

Základem jsou opět dva směry, pro $v_1 = (1)$, respektive $v_2 = (-1)$. Vztah (5.9) dosazením za vektor v zjednodušíme. Dodejme, že podmínka musí být splněna pro oba směry současně.

$$\partial f(0) = \{\zeta \in \mathbb{R} \mid f^0(0{,}1) \geq \langle \zeta, 1\rangle\} \cap \{\zeta \in \mathbb{R} \mid f^0(0,-1) \geq \langle \zeta, -1\rangle\}. \tag{5.10}$$

Nyní již počítejme konkrétně pro každou funkci. Pro funkci $f_1 = x^2$ jsou v bodě $x = 0$ obě směrové derivace shodné, dosadíme je do vztahu (5.10)

$$\begin{gathered} f_1^0(0,-1) = f_1^0(0{,}1) = 0, \\ \partial f_1(0) = \{\zeta \in \mathbb{R} \mid 0 \geq \langle \zeta, 1\rangle\} \cap \{\zeta \in \mathbb{R} \mid 0 \geq \langle \zeta, -1\rangle\}. \end{gathered} \tag{5.11}$$

Vztah (5.11) můžeme přepsat na soustavu nerovnic

$$\begin{gathered} 0 \geq 1 \cdot \zeta \iff \zeta \leq 0, \\ 0 \geq -1 \cdot \zeta \iff \zeta \geq 0. \end{gathered}$$

Jediné řešení soustavy je $\zeta = 0$ a tedy $\boldsymbol{\partial f_1(0) = \{0\}}$.

Přejděme k funkci $f_2 = |x|$. Postupujeme totožně.

$$\begin{gathered} f_2^0(0,-1) = f_2^0(0{,}1) = 4, \\ \partial f_2(0) = \{\zeta \in \mathbb{R} \mid 4 \geq \langle \zeta, 1\rangle\} \cap \{\zeta \in \mathbb{R} \mid 4 \geq \langle \zeta, -1\rangle\}. \end{gathered} \tag{5.12}$$

Vztah (5.12) můžeme přepsat na soustavu nerovnic

$$\begin{gathered} 4 \geq 1 \cdot \zeta \iff \zeta \leq 4, \\ 4 \geq -1 \cdot \zeta \iff \zeta \geq -4. \end{gathered}$$

Soustava má řešení je $\zeta \in [-4;\ 4]$ a tedy $\boldsymbol{\partial f_2(0) = [-4;\ 4]}$.

△

Věta 5.13: Vztah mezi Clarkeovou směrovou derivací a zobecněným gradientem

Nechť funkce $f : \mathbb{R}^n \to \mathbb{R}$ je lipschitzovsky spojitá v okolí bodu x. Pak platí

$$\forall v \in \mathbb{R}^n:\ f^0(x, v) = \max\{\langle \zeta, v \rangle \mid \zeta \in \partial f(x)\}$$

Věta (5.13) nám říká, že Clarkeova směrová derivace je rovna největšímu ze skalárních součinů $\langle \zeta, v \rangle$, kde ζ jsou Clarkeovy subgradienty a v je směrový vektor.

Výpočet zobecněného gradientu podle definice není příliš efektivní. Ukážeme si jiný způsob výpočtu.

Věta 5.14: Alternativní výpočet Clarkeova zobecněného gradientu

Nechť funkce $f : \mathbb{R}^n \to \mathbb{R}$ je lipschitzovsky spojitá v okolí bodu x. Pak platí

$$\partial f(x) = \operatorname{conv}\left\{\lim_{i\to\infty} \nabla f(x_i) \mid x_i \to x, x_i \notin \Omega_f\right\}$$

Kde $\Omega_f = \{x \in \mathbb{R}^n \mid f \text{ není diferencovatelná v } x\}$

Výpočet zobecněného gradientu podle věty (5.14) pro lokálně lipschitzovsky spojité funkce je umožněn díky Rademacherově větě (5.4), která zaručuje diferencovatelnost funkce téměř všude. Konkrétně pro dimenzi $n = 1$ tvoří množinu Ω_f jednotlivé body, pro $n = 2$ jednotlivé body nebo křivky, v případě $n = 3$ navíc ještě plochy. Lebesguova míra těchto množin je pro danou dimenzi vždy nulová. Pro lepší pochopení věty (5.14) postupujme následovně.

$$M = \left\{\lim_{i\to\infty} \nabla f(x_i) \mid x_i \to x, x_i \notin \Omega_f\right\}, \tag{5.15}$$

$$\partial f(x) = \operatorname{conv}(M). \tag{5.16}$$

Nejdříve se zaměřme na množinu M. Přibližujme se k bodu x z různých směrů daných osami všech nezávislých proměnných. Pak určitě existují v bodě x obě jednostranné derivace (v daném směru), tyto spočítejme. Po výpočtu pro všechny osy (respektive směry jimi určenými) v daném bodě x získáme 2^n diferenciálů. Shodné diferenciály můžeme vyřadit (ale nemusíme, výsledek bude stejný). Tím získáme množinu vektorů M, která má nejvýše 2^n prvků (n je dimenze dané úlohy). Konvexní obal množiny M je podmnožinou Clarkeova zobecněného gradientu. Pro dimenzi $n = 1$ lze postup zjednodušit. Množinu M tvoří $2^1 = 2$ vektory – jednostranné derivace v bodě x, tedy

$$M = \left\{\left(\frac{\partial f}{\partial x^+}\right);\ \left(\frac{\partial f}{\partial x^-}\right)\right\}. \tag{5.17}$$

Pro dimenzi $n = 2$ je $|M| = 4$, tvoří ji následující diferenciály (které získáme jako kombinace jednostranných derivací podle různých proměnných)

$$M = \left\{\left(\frac{\partial f}{\partial x^+}; \frac{\partial f}{\partial y^+}\right); \left(\frac{\partial f}{\partial x^+}; \frac{\partial f}{\partial y^-}\right); \left(\frac{\partial f}{\partial x^-}; \frac{\partial f}{\partial y^+}\right); \left(\frac{\partial f}{\partial x^-}; \frac{\partial f}{\partial y^-}\right)\right\}. \tag{5.18}$$

Stejně postupujeme pro vyšší dimenze. Zopakujme, že opakující se vektory (diferenciály) z množiny M můžeme ale nemusíme vyřadit, výsledek to neovlivní. *„Vyřadit"* představuje jistou výpočetní náročnost navíc, stejně tak jako výpočet se zbytečně velkým počtem vektorů. Záleží na konkrétním zadání, která varianta bude rychlejší.

Věta 5.19: Clarkeovy stacionární body

Nechť funkce $f : \mathbb{R}^n \to \mathbb{R}$ je lokálně lipschitzovsky spojitá na $\mathbb{R}^n$. Jestliže funkce f nabývá v bodě x svého lokálního maxima nebo minima, pak platí

$$o \in \partial f(x)$$

Body splňující tuto podmínku nazýváme *Clarkeovy stacionární body* optimalizační úlohy

$$\min_{x \in \mathbb{R}^n} f(x)$$

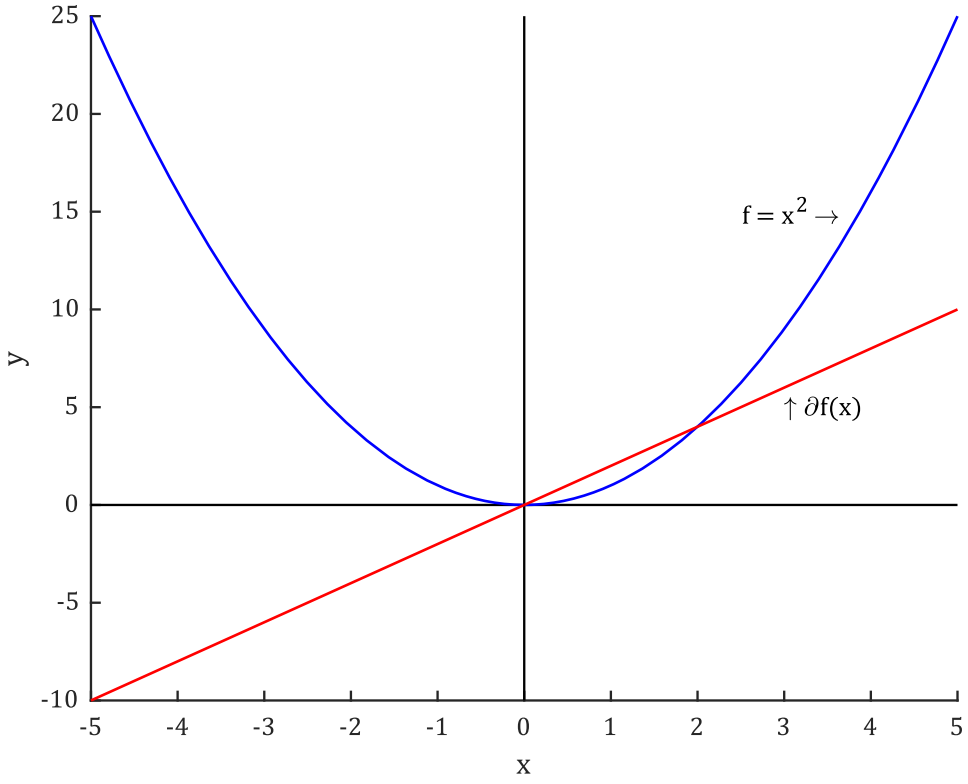
Věta (5.19) říká, že pokud v bodě x nastává lokální extrém, pak Clarkeův zobecněný gradient obsahuje nulový vektor o (jedním ze subgradientů je nulový vektor). Pro dimenzi $n = 1$ lze psát $0 \in \partial f(x)$.

Příklad 5.20: Graficky znázorněte Clarkeův zobecněný gradient funkcí $f_1 = x^2$ a $f_2 = |4x|$ na celém definičním oboru. V bodech $x_1 = 0, x_2 = 3, x_2 = -3$ jej vypočtěte podle věty (5.14). Ověřte výsledky podle věty (5.13) a možnost lokálních extrémů podle věty (5.19). Použijte výsledky z příkladu (5.7).

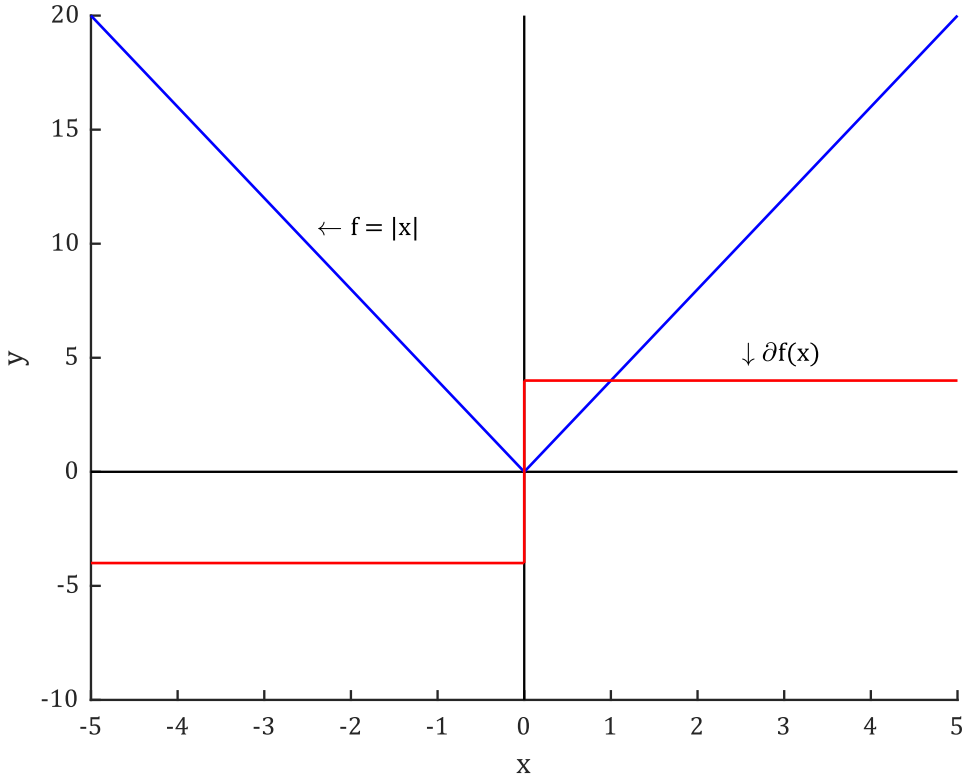
Řešení: Zobecněný gradient již známe, můžeme vykreslit oba grafy, *Obrázky 9 a 10*. Všimněme si zobecněného gradientu funkce $f_2 = |x|$ v bodě $x = 0$, tedy $\partial f_2(0)$. Nejedná se o konkrétní hodnotu, ale o interval $\partial f_2(0) = [-4; 4]$. Dále provedeme výpočet podle věty (5.14), avšak vynechejme veškeré úvahy o hladké funkci – počítejme, jako kdybychom vůbec nevěděli, zdali je funkce v daných bodech hladká. Nejdříve stanovme množiny M podle (5.17) pro každý bod a funkci.

$M = \left\{\left(\frac{\partial f}{\partial x^+}\right); \left(\frac{\partial f}{\partial x^-}\right)\right\}$	$f_1 = x^2$	$f_2 = \|x\|$
$x_1 = -3$	$\{-6; -6\}$	$\{-4; -4\}$
$x_2 = 0$	$\{0; 0\}$	$\{+\mathbf{4}; -\mathbf{4}\}$
$x_3 = +3$	$\{+6; +6\}$	$\{+4; +4\}$

Tabulka 5 – K příkladu 5.20, stanovení množin.



Obrázek 9 – Příklad 5.20, Clarkeův zobecněný gradient hladké funkce.



Obrázek 10 – Příklad 5.20, Clarkeův zobecněný gradient nehladké funkce.

Opět si všimněme, že pouze u funkce f_2 se množina M skládá z různých vektorů, ostatní množiny můžeme zjednodušit na jednoprvkové. Pro ukázku výpočtu je ponechme v původním tvaru. Konvexní obal množiny vektorů vypočítáme následovně.

$$\mathrm{conv}\{v_1, v_2, \dots, v_n\} = \left\{ \sum_{i=1}^{n} a_i v_i \;\middle|\; \sum_{i=1}^{n} a_i = 1 \ \wedge\ \forall i\colon a_i \geq 0 \right\}. \tag{5.21}$$

Konvexní obal je nejmenší konvexní množina, která obsahuje všechny vektory $v_1, v_2, \dots, v_n$. Nyní už je zřejmé, proč odebrání stejných vektorů z množiny M konvexní obal nezmění. Nechť nejdříve množina M obsahuje pouze totožné vektory

$$M = \{v_1, v_2, \dots, v_n \,|v = \ v_1 = v_2 = \cdots = v_n\},$$

$$\mathrm{conv}(M) = \{a_1 v_1 + a_2 v_2 + \cdots + a_n v_n |\ a_1 + a_2 + \cdots + a_n = 1 \ \wedge\ \forall i\colon a_i \geq 0\},$$

$$a_1 v_1 + a_2 v_2 + \cdots + a_n v_n = a_1 v + a_2 v + \cdots + a_n v = v(a_1 + a_2 + \cdots + a_n) = v \cdot 1 = v.$$

Proto Clarkeův zobecněný gradient v libovolném bodě funkce f_1 a s výjimkou bodu $x = 0$ i funkce f_2 je roven (obyčejnému) gradientu, v naší úloze (obyčejné) derivaci. Dořešme zobecněný gradient $\partial f_2(0)$.

$$M = \{4, -4\},$$

$$\mathrm{conv}(M) = \{4a_1 - 4a_2 |\ a_1 + a_2 = 1 \ \wedge\ \forall i\colon a_i \geq 0\}.$$

Jak dál? Ze vztahu $a_1 + a_2 = 1$ plyne $a_2 = 1 - a_1$, dosadíme do $4a_1 - 4a_2$ a získáme (H_h značí obor hodnot funkce h). Výsledky poté shrneme do *Tabulky 8.*

$$h = 4(a_1 - a_2) = 4(a_1 - 1 + a_1) = 8a_1 - 4; a_1 \in [0; 1] \Longrightarrow\ H_h = [\mathbf{-4};\ \mathbf{+4}] = \boldsymbol{\partial f_2(0)}.$$

$\boldsymbol{\partial f(x)}$	$f_1 = x^2$	$f_2 = \lvert x \rvert$
$x_1 = -3$	$0 \notin \{-6\}$	$0 \notin \{-4\}$
$x_2 = 0$	$\mathbf{0} \in \{\mathbf{0}\}$	$\mathbf{0} \in [\mathbf{-4};\ \mathbf{+4}]$
$x_3 = +3$	$0 \notin \{+6\}$	$0 \notin \{+4\}$

Tabulka 6 – Řešení příkladu 5.20 - Clarkeův zobecněný gradient, množinový zápis. Nulový vektor je v $\mathbb{R}$ *reprezentován o* = (0).

Zkontrolujeme získané výsledky z hlediska věty (5.13), stručně zopakujme

$$\forall v \in \mathbb{R}^n\colon\ f^0(x, v) = \max\{\ \langle \zeta, v \rangle \mid \zeta \in \partial f(x)\}. \tag{5.22}$$

Postupně vyplňujeme následující tabulku. Pro f_1 stačí pouze dosadit, pro f_2 rovněž s výjimkou $x = 0$.

$$f_1^0(-3,-1) = \max\{\langle -6,-1\rangle\} = \max\{+6\} = +6,$$
$$f_1^0(-3,+1) = \max\{\langle -6,+1\rangle\} = \max\{-6\} = -6.$$

Dále pokračujeme stejně až k položce $f_2^0(0,v)$, jejíž výpočet si ukážeme pro oba směry najednou:

$$f_2^0(0,\pm 1) = \max\{\langle \zeta,\pm 1\rangle \mid \zeta \in \partial f_2(0)\},$$

$$h = \langle \zeta,\pm 1\rangle = \pm 1 \cdot a = \pm a; a \in [-4;4] \implies H_h = [-4;4],$$

$$f_2^0(0,\pm 1) = \max(H_h) = +4.$$

$\boldsymbol{f^0(x,v)}$	$f_1 = x^2$		$f_2 = \lvert x\rvert$	
	$v = (-1)$	$v = (1)$	$v = (-1)$	$v = (1)$
$x_1 = -3$	$+6$	-6	$+4$	-4
$x_2 = 0$	0	0	$+4$	$+4$
$x_3 = +3$	-6	$+6$	-4	$+4$

Tabulka 7 – Řešení příkladu 5.20, o*věření vztahu mezi Clarkeovou směrovou derivací a zobecněným gradientem.*

Výsledky jsou v pořádku, můžeme porovnat s *Tabulkou 1* pro f_1 a *Tabulkou 3* pro f_2. Zbývá dořešit poslední část úlohy, a to použití věty (5.19). Ta říká, že pokud funkce f má v bodě x lokální extrém, pak nulový vektor patří do zobecněného gradientu, tedy $o \in \partial f(x)$. Řešení přímo plyne z *Tabulky 6* pro bod $x = 0$, ve kterém mají obě funkce ostré lokální minimum.

△

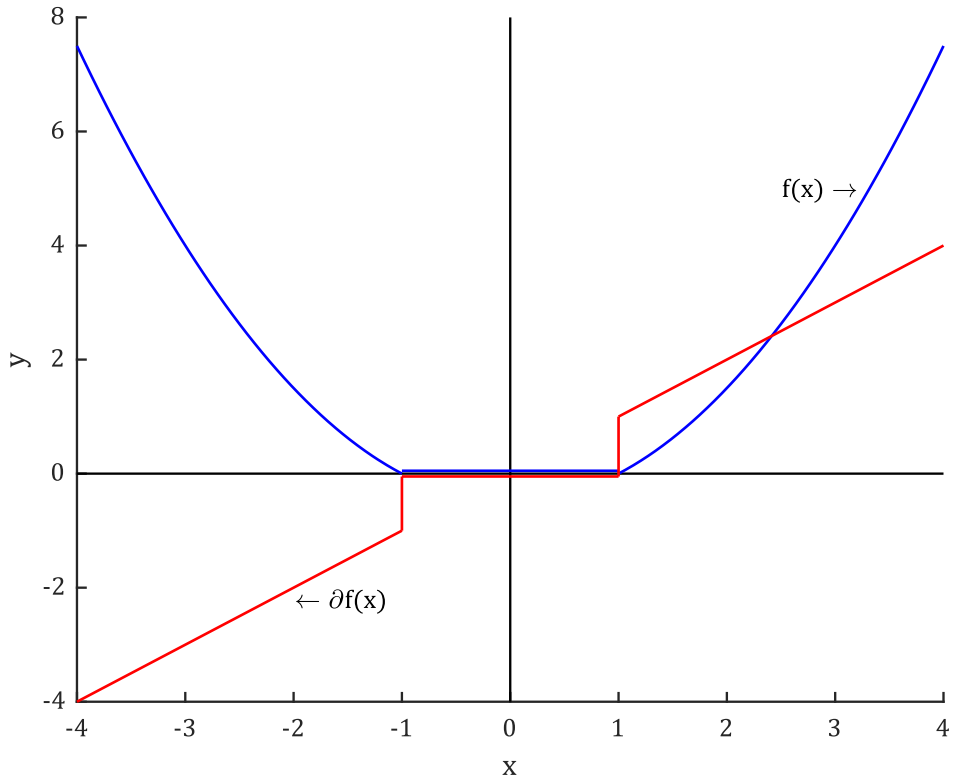
Příklad 5.23: Určete Clarkeův zobecněný gradient a stacionární body funkce

$$f(x) = \max\left\{0; \frac{1}{2}(x^2-1)\right\}.$$

Řešení: Standardně nejdříve vykreslíme graf. Následně vypočteme zobecněný gradient a zjistíme, ve kterých bodech $x \in \mathbb{R}$ platí $0 \in \partial f(x)$. Konkrétně f můžeme přepsat do tvaru

$$f(x) = \begin{cases} \frac{1}{2}(x^2-1); & x \in (-\infty;-1), \\ 0; & x \in \langle -1;1\rangle, \\ \frac{1}{2}(x^2-1); & x \in (1;\infty). \end{cases}$$

Nyní vidíme, že funkce je nehladká v bodech $x_1 = -1$ a $x_2 = 1$. V těchto bodech se zobecněný gradient liší od běžného gradientu. Vypočteme jej podle věty (5.14), určíme nejdříve diferenciál funkcí.



Obrázek 11 – Příklad 5.23, Graf „max funkce" a Clarkeova zobecněného gradientu.

$$\nabla f(x) = \begin{cases} x; & x \in (-\infty;\ -1), \\ 0; & x \in (-1; 1), \\ x; & x \in (1;\ \infty). \end{cases}$$

Pomocí diferenciálu určíme zobecněný gradient

$$\boldsymbol{\partial f(x)} = \begin{cases} x; & x \in (-\infty;\ -1), \\ [-1; 0]; & x = -1, \\ 0; & x \in (-1; 1), \\ [0; 1]; & x = 1, \\ x; & x \in (1;\ \infty). \end{cases}$$

Clarkeovy stacionární body tvoří interval $[-\mathbf{1}; \mathbf{1}]$, ve všech bodech intervalu platí $o \in \partial f(x)$.

△

5.3 Numerický výpočet zobecněného gradientu

V minulé kapitole jsme probrali zobecněný gradient a jeho složky – subgradienty. Úlohu jsme ovšem umístili do jednorozměrného prostoru, navíc jsme uvažovali v podstatě triviální funkci f. Tím jsme úlohu maximálně zjednodušili, a přesto byl výpočet značně náročný. Nyní se zaměříme na běžné zadání funkce f (vyplývající z praxe) a dále budeme předpokládat obecnou dimenzi úlohy. V praxi se můžeme setkat převážně se dvěma druhy zadání cenové funkce. Jedno jsme již řešili v úloze (5.23) – takové zadání se nazývá ***„max funkce“***. Druhou možností zadání je lineární kombinace s nezápornými koeficienty lineárních a kvadratických konvexních funkcí, nazvěme jej ***„suma funkce“***. Oba *výsledné* typy funkcí zachovávají *konvexitu* i *hladkost téměř všude*.

Max funkce	**Suma funkce**
$\boldsymbol{f(x) = \max_{i=1,\dots,k} f_i(x)}$ $x \in \mathbb{R}^n$ f_i jsou konvexní	$\boldsymbol{f(x) = \sum_{i=1}^{k} a_i f_i(x)}$ $a_i \geq 0; x \in \mathbb{R}^n$ f_i jsou konvexní

Tabulka 8 – Přehled obvyklého zadání nehladkých cenových funkcí.

U takovýchto funkcí, navíc v n-rozměrném prostoru, je analytický výpočet zobecněného i běžného gradientu prakticky vyloučen. Musíme použít numerický výpočet. Proč jej uvádíme až teď?

Při numerickém výpočtu nikdy nevíme, zdali jsme se ocitli v bodě, kde je funkce hladká nebo opačně. Numerickým výpočtem to zaručeně nezjistíme, protože směrovou derivaci aproximujeme, obvykle lineárním modelem, kdy výpočet provádíme pomocí dalších bodů v okolí bodu x_0, kde chceme derivaci spočítat. Uvnitř tohoto okolí však nevíme (zanedbáváme), jak se funkce chová. A takové okolí bodu x_0 existuje vždy. Uvažujme výpočet směrové derivace v bodě x_0 pomocí levého i pravého okolí x_0. V případě hladké funkce jsme hodnoty zprůměrovali. Pro nehladkou funkci si ponecháme obě hodnoty – místo konkrétní hodnoty budeme uvažovat interval. To je jediný rozdíl.

Nyní formálněji. Jaké směry pro směrové derivace volíme? Použít můžeme libovolnou ortonormální množinu n vektorů, v praxi je vhodné použít jednotkové vektory ve směru kladných částí os nezávislých proměnných. A v těchto směrech v bodě x_0 vypočteme derivace pomocí levého i pravého okolí. Požadovanou přesnost volíme parametrem $\zeta > 0$, směr značíme v,

$$f'_+(x, v) = \frac{1}{\zeta}[f(x + \zeta v) - f(x)], \tag{5.25}$$

$$f'_{-}(x,v) = \frac{1}{\zeta}[f(x) - f(x - \zeta v)f]. \tag{5.26}$$

Uvažujme množinu jednotkových vektorů tvořících ortonormální bázi prostoru $\mathbb{R}^n$, $\{e_1, e_2, \dots, e_n\}$, směr každého z nich je dán směrem kladné části příslušné osy nezávislé proměnné. Pak

$$\partial f(x) \supseteq \text{conv}\{[f'_{-}(x,e_1), f'_{+}(x,e_1)];\ [f'_{-}(x,e_2), f'_{+}(x,e_2)]; \dots; [f'_{-}(x,e_n), f'_{+}(x,e_n)]\}. \tag{5.27}$$

Ze všech možných směrů (tvořících kompaktní konvexní množinu) potřebujeme vybrat jediný. Podobně jako u hladké funkce, za směr poklesu volíme záporně vzatý zobecněný gradient (avšak narozdíl od hladké funkce, záporně vzatý zobecněný gradient nemusí představovat skutečný směr poklesu). Pro hledání směru, kterým se při hledání minima vydáme, ovšem neexistuje kritérium určující ten nejlepší. Proto následující algoritmus zvolíme stejně, jako kdybychom počítali běžný gradient. Algoritmus je implementován v Příloze C1 a je použitelný pro hladkou i nehladkou funkci.

Algoritmus 5.28: ***Numerický výpočet (Clarkeova zobecněného) gradientu***

1. **Vstup:** $f : \mathbb{R}^n \to \mathbb{R}$, $x_0 \in \mathbb{R}^n$, $\zeta > 0$, n
2. **Inicializace**

   g je nulový sloupcový vektor délky n

   $E \coloneqq I^{n\times n}$ (jednotková matice, s_i^E značí i-tý sloupec matice E)
3. **Iterační cyklus**

   for $i = 1 : n$

   $$g(i) \coloneqq \frac{1}{2\zeta}\left(f\left(x_0 + \zeta s_i^E\right) - f\left(x_0 - \zeta s_i^E\right)\right)$$

   end for
4. **Výstup:**

   g (gradient, respektive jeden z možných gradientů)

5.4 Závěrečné shrnutí

- **Důležitou vlastností cenové funkce je její lokální lipschitzovská spojitost.** Jedná se o silnější vlastnost než *pouhá* spojitost funkce.
- Ukázali jsme způsob **výpočtu Clarkeovy směrové derivace a Clarkeova zobecněného gradientu.** Obvykle můžeme zvolit libovolný vektor z množiny $\partial f(x)$. Volba směru tedy vychází ze stejného principu jako u minimalizace hladké funkce. V případě varianty vycházející z metody CG musíme použít o něco obecnější způsob výpočtu transformační matice (metoda se nazývá SDG).

6 Minimalizace nediferencovatelné funkce

Hned na úvod poznamenejme, že se budeme zabývat minimalizací pouze *téměř všude hladkých funkcí*, konkrétně *lokálně lipschitzovsky spojitých* funkcí. Jak jsme již uvedli v předcházejících kapitolách, existují dva základní způsoby minimalizace – a to **přímé a nepřímé metody.** Naši pozornost budeme věnovat prakticky výhradně metodám přímým. Obecnější znalosti nám poskytne literatura [1] a [9].

Obecně při minimalizaci potřebujeme určit

- **Směr poklesu** η
- **Délku kroku** h
- **Ukončovací kritérium algoritmu**
- **Iterační předpis** ve tvaru

$$x_{k+1} = x_k + \eta_k h_k. \tag{6.1}$$

Určení směru poklesu je podobné jako u gradientních metod (numericky může být dokonce totožné), pouze gradient nahradíme subgradientem. **Určení délky kroku** představuje mnohem složitější problém. Nejjednodušší řešení spočívá ve stanovení délky kroku předem (a-priori), ale současně se jedná rovněž o řešení nejpomalejší. Stanovení **ukončovacích podmínek** výpočtu také není triviální, často používáme několik kritérií spojených logickou spojkou *nebo*. Než se pustíme do přímých optimalizačních metod, krátce si představíme metody nepřímé.

6.1 Nepřímé optimalizační metody

Princip nepřímých metod spočívá v nahrazení nehladké funkce funkcí hladkou, a následné minimalizaci pomocí algoritmů pro hladké funkce.

6.1.1 Interpolace a aproximace

Nehladkou funkci můžeme nejdříve interpolovat nebo aproximovat funkcí hladkou a následně použít metody pro minimalizaci hladké funkce.

Interpolací funkce rozumíme interpolaci funkčních hodnot v uzlech sítě bodů (nezávislé proměnné) tak, aby rozdíl funkčních hodnot interpolační a interpolované funkce v uzlech sítě byl co nejnižší. Uvedený postup se hodí zejména v případě, kdy máme zadány výsledky měření v podobě izolovaných hodnot.

V případě nehladké optimalizace bychom zvolili lineární interpolaci. Pro potřeby hladké optimalizace můžeme zvolit dvě řešení – buďto stupeň interpolačního polynomu odpovídající počtu měření sníženého o jedna, anebo interpolaci kubickým splinem. První metoda je pro vyšší počet měření časově náročná a často vede na špatně podmíněnou matici soustavy rovnic koeficientů interpolačního polynomu. Druhá možnost, kubický spline, pro m měření zase vytvoří $m - 1$ funkcí.

Aproximací funkce rozumíme nahrazení původní funkce novou tak, aby velikost rozdílu funkčních hodnot obou funkcí byla v jistém smyslu co nejmenší. Původní funkci nahrazujeme často polynomiální funkcí, stupeň polynomu volíme podle potřeby. Velikost rozdílu je určena metrikou, tu opět můžeme zvolit libovolně, avšak vhodně vzhledem k povaze řešené úlohy. Uvedený postup má opět řadu úskalí.

Mezi hlavní nevýhody nelineární interpolace a aproximace patří

- Výpočetní náročnost pro vyšší stupeň interpolačního a aproximačního polynomu,
- Velké zkreslení při nižším stupni polynomu,
- Zvlnění cenové funkce při vyšším stupni polynomu a tím vznik mnoha dalších lokálních extrémů,
- Aproximace i interpolace vedou na Van der Mondovu matici, která je často velmi špatně podmíněná, a tím opět vzniká velké zkreslení,

Poznámka – **lineární interpolaci** velmi často používáme při mnoha výpočtech, aniž bychom si to vůbec uvědomili. Třeba numerický výpočet derivace vycházející z lineárního modelu je typickým příkladem.

6.1.2 Nepřímá metoda s penalizací

Jedná se o jednoduchou metodu řešící pouze úzký okruh minimalizačních úloh. Cenová funkce má typický suma zápis absolutních hodnot, který je pro tuto metodu typický. Řešíme tedy úlohu

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{k} |f_i(x)| \,, \tag{6.2}$$

kde funkce f_i jsou spojitě diferencovatelné. Pomocí **penalty** $\varepsilon > 0$ úlohu (6.2) převedeme na hladkou úlohu

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{k} (f_i(x)^2 + \varepsilon)^{\frac{1}{2}} \,. \tag{6.3}$$

Dále postupujeme stejně jako u hladké funkce. Problém spočívá v nutnosti volby penalty předem, stejné pro všechny funkce f_i, přičemž pro každou funkci by byla ideální jiná penalta. Tato metoda v praxi nemá příliš velké využití.

6.2 Přímé optimalizační metody

Přímé metody vynechávají interpolační nebo aproximační krok, což výpočet obvykle urychlí a současně zpřesní. Musíme ovšem upravit minimalizační algoritmus – největší problém představuje určení délky kroku a zastavení výpočtu.

6.2.1 Operátor dilatace prostoru SD

Než si ukážeme různé možnosti výpočtu délky kroku, musíme probrat dilataci prostoru (anglicky space dilation – SD). Ta se používá pro metody podobné sdruženým gradientům, jedná se vlastně o obecnější výpočet transformační matice. Pro další úvahy a výpočty si zjednodušme život a předpokládejme, že souřadnice všech vektorů prostoru $\mathbb{R}^n$ jsou vyjádřeny v **ortonormální bázi** tohoto prostoru.

Nechť je dán jednotkový směrový vektor $\xi \in \mathbb{R}^n, \|\xi\| = 1$ a konstantní číslo $\alpha \geq 0$. Pak každý vektor $x \in \mathbb{R}^n$ lze zapsat ve tvaru (uvažujme vše jako sloupcové vektory)

$$x = \gamma_\xi(x)\xi + d_\xi(x), \tag{6.4}$$

$$\text{kde } \langle \xi, d_\xi(x) \rangle = 0, \tag{6.5}$$

$d_\xi(x) \in \mathbb{R}^n$ a $\gamma_\xi(x) \in \mathbb{R}$. Potřebujeme získat koeficient $\gamma_\xi(x)$. Ze vztahu (6.5) přímo plyne, že hledaný vektor $d_\xi(x)$ je kolmý na vektor ξ. Protože ξ je jednotkový vektor, platí $\xi^T\xi = \langle \xi, \xi \rangle = 1$. Vztah (6.4) postupně upravme.

$$x = \gamma_\xi(x)\xi + d_\xi(x) \quad | \cdot \xi^T \; zleva,$$

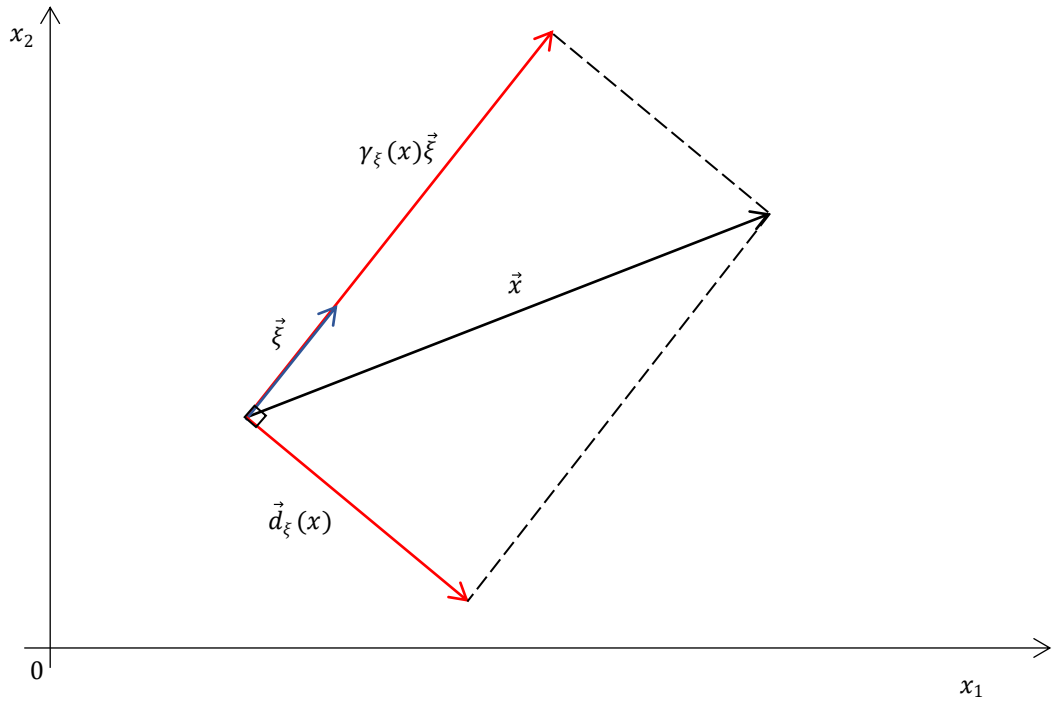
$$\xi^T x = \xi^T \gamma_\xi(x)\xi + \xi^T d_\xi(x) = \gamma_\xi(x)\langle \xi, \xi \rangle + \langle \xi, d_\xi(x) \rangle = \gamma_\xi(x) \cdot 1 + 0 = \gamma_\xi(x),$$

$$\gamma_\xi(x) = \xi^T x = \langle x, \xi \rangle. \tag{6.6}$$

Vztah (6.6) nám říká, jak získat koeficient $\gamma_\xi(x)$. Koeficient dosadíme do (6.4) a získáme vektor $d_\xi(x)$

$$d_\xi(x) = x - \langle x, \xi \rangle \xi. \tag{6.7}$$

Grafické znázornění rozkladu vektoru x ve tvaru (6.4) je nakresleno na *Obrázku 12*. Jedná se o rozklad vektoru x do směrů ξ a $d_\xi(x)$, $d_\xi(x) \perp \xi$. Je zřejmé, že všechny tři vektory musí ležet v jedné rovině, koeficient $\gamma_\xi(x)$ a vektor $d_\xi(x)$ jsou pro daný směr ξ určeny jednoznačně. Později za směr ξ dosadíme subgradient.



Obrázek 12 – Rozklad vektoru.

Definice 6.8: Operátor dilatace prostoru

Operátor $R_\alpha(\xi)$, který transformuje vektor x ve tvaru (6.4) na tvar

$$R_\alpha(\xi)x = \alpha\gamma_\xi(x)\xi + d_\xi(x)$$

se nazývá operátor dilatace prostoru ve směru vektoru ξ s koeficientem α.

Operátor $R_\alpha(\xi)$ je (při našem zjednodušení) čtvercová symetrická matice velikost $n \times n$, můžeme ji rovněž nazvat maticí lineární transformace. Existuje pro něj několik velice užitečných tvrzení včetně způsobu, jak je získáme. Nejprve si všechna tato tvrzení uvedeme a posléze některé z nich dokážeme.

Věta 6.9: Tvrzení týkající se operátoru dilatace prostoru

Vycházejme ze vztahů (6.4-6.7) a definice (6.8). Pak následující tvrzení jsou pravdivá:

i. $R_\alpha(\xi)x = (\alpha - 1)\langle x, \xi\rangle\xi + x$

ii. Operátor $R_\alpha(\xi)$ je lineární a symetrický (tedy je to symetrická matice)

iii. $R_{\alpha\beta}(\xi) = R_\alpha(\xi)R_\beta(\xi)$

iv. Pro každé $\alpha > 0$ platí $R_\alpha(\xi)R_{1/\alpha}(\xi) = R_1(\xi) = I$, I je jednotková matice.

v. Operátor $R_0(\xi)$ zobrazí vektor x na vektor kolmý k ξ

$$R_0(\xi)x = d_\xi(x)$$

vi. Nechť $s = \{e_1, e_2, \dots, e_n\}$ je ortonormální báze prostoru $\mathbb{R}^n$ a vektor ξ má v této bázi souřadnice $\xi = (\xi_1, \xi_2, \dots, \xi_n)$. Potom z (i, ii) plyne, že operátor $R_\alpha(\xi)$ je symetrická matice, jejíž prvky $\{r_{ij}\}$ jsou definovány následovně:

$$r_{ij} = \langle R_\alpha(\xi)e_i, e_j\rangle = \begin{cases}(\alpha-1)\xi_i\xi_j^T & pro\ i \neq j,\\ (\alpha-1)\xi_i\xi_i^T + 1 & pro\ i = j.\end{cases}$$

vii. Operátor $R_\alpha(\xi)$ má pro $n \geq 2$ dvě vlastní čísla, a to $\lambda_1 = \alpha$ a $\lambda_2 = 1$. První vlastní číslo odpovídá podprostoru vlastních vektorů generovaných směrovým vektorem ξ, druhé podprostoru vlastních vektorů ortogonálních k ξ.

viii. Výpočet vektoru $R_\alpha(\xi)x$ vyžaduje $2n + 1$ násobení, výpočet matice ve tvaru $R_\alpha(\xi)A$ respektive $AR_\alpha(\xi)$ pak $n(2n + 1)$ násobení.

ix. Pro libovolný vektor $x \in \mathbb{R}^n$ platí

$$\|R_\alpha(\xi)x\| = \sqrt{\|x\|^2 + (\alpha^2 - 1)\langle x, \xi\rangle^2}$$

x. Operátor $R_\alpha(\xi)$ můžeme maticově zapsat jako

$$R_\alpha(\xi) = I + (\alpha - 1)\xi\xi^T$$

Důkaz:

(i) Tento vztah je pro nás výhodný, nevyžaduje totiž výpočet parametrů $\gamma_\xi(x)$ a $d_\xi(x)$. Do (6.8) dosadíme vztahy (6.6-7) a získáme $R_\alpha(\xi)x = \alpha\langle x, \xi\rangle\xi + x - \langle x, \xi\rangle\xi = (\alpha - 1)\langle x, \xi\rangle\xi + x$.

(ii) $\langle R_\alpha(\xi)x, y\rangle = \langle(\alpha - 1)\langle x, \xi\rangle\xi + x, y\rangle = (\alpha - 1)\langle x, \xi\rangle\xi^T y + x^T y = (\alpha - 1)\langle x, \xi\rangle\langle y, \xi\rangle + \langle x, y\rangle = (\alpha - 1)\langle y, \xi\rangle\langle x, \xi\rangle + \langle x, y\rangle = \langle(\alpha - 1)\langle y, \xi\rangle\xi + y, x\rangle = \langle R_\alpha(\xi)y, x\rangle$.

(iii) Nejdříve dokážeme tvrzení (x), následně už je důkaz (iii) snadný.

$$R_{\alpha\beta}(\xi)x = R_\beta(\xi)R_\alpha(\xi)x = (I + (\beta - 1)\xi\xi^T)(I + (\alpha - 1)\xi\xi^T) =$$
$$= I^2 + I(\alpha - 1)\xi\xi^T + I(\beta - 1)\xi\xi^T + (\beta - 1)\xi\xi^T(\alpha - 1)\xi\xi^T =$$
$$= I^2 + I(\beta - 1)\xi\xi^T + I(\alpha - 1)\xi\xi^T + (\alpha - 1)\xi\xi^T(\beta - 1)\xi\xi^T =$$
$$= (I + (\alpha - 1)\xi\xi^T)(I + (\beta - 1)\xi\xi^T) = R_\alpha(\xi)R_\beta(\xi) = R_{\beta\alpha}(\xi)x.$$

(iv) $R_\alpha(\xi)R_{1/\alpha}(\xi) = R_{\alpha\cdot 1/\alpha}(\xi) = R_1(\xi) \implies R_1(\xi)x = (1 - 1)\langle x, \xi\rangle\xi + x = x \implies R_1(\xi) = I$

(v) $R_0(\xi)x = 0\gamma_\xi(x)\xi + d_\xi(x) = d_\xi(x)$

(ix) $\|R_\alpha(\xi)x\|^2 = \langle R_\alpha(\xi)x, R_\alpha(\xi)x\rangle = \langle(\alpha-1)\langle x,\xi\rangle\xi + x, (\alpha-1)\langle x,\xi\rangle\xi + x\rangle =$
$(\alpha-1)^2\langle x,\xi\rangle^2\|\xi\|^2 + 2(\alpha-1)\langle x,\xi\rangle\langle\xi,x\rangle + \|x\|^2 = (\alpha-1)^2\langle x,\xi\rangle^2 + 2(\alpha-1)\langle x,\xi\rangle^2 +$
$+\|x\|^2 = \langle x,\xi\rangle^2(\alpha^2 - 2\alpha + 1 + 2\alpha - 2) + \|x\|^2 = (\alpha^2-1)\langle x,\xi\rangle^2 + \|x\|^2$

(x) $R_\alpha(\xi)x = (I + (\alpha-1)\xi\xi^T)x = x + (\alpha-1)\xi\langle\xi,x\rangle = (\alpha-1)\langle\xi,x\rangle\xi + x$, viz (i). Pozor, součin $\xi\xi^T$ je matice $n \times n$, nikoli 1.

■

6.2.2 Délka kroku daná předem

Dá se dokázat, že při splnění vztahů (6.11) a (6.12) pro délku kroku h subgradientní metody konvergují ke správnému řešení.

$$\lim_{k\to\infty} h_k = 0, \tag{6.11}$$

$$\sum_{i=0}^{k} h_k = \infty. \tag{6.12}$$

Nejznámější předpis pro délku kroku představuje harmonická posloupnost

$$h_k = \frac{c}{k+1}; \quad c \in \mathbb{R}^+, k = 0,1,2,\ldots \tag{6.13}$$

Tím získáme **iterační předpis pro minimalizaci konvexní funkce** (η_k je uvažovaný směr poklesu)

$$\boldsymbol{x_{k+1} = x_k + \eta_k \cdot \frac{c}{k+1}}. \tag{6.14}$$

Algoritmus 6.15: ***Výpočet nové aproximace pro délku kroku danou předem***

i. **Vstup:** Aktuální aproximace x_k, normovaný směr poklesu η, počáteční délka kroku c, index aktuální iterace k

ii. **Výpočet:**

$$x_{k+1} \coloneqq x_k + \eta \cdot \frac{c}{k+1}$$

iii. **Výstup:**

x_{k+1}, h_k

Algoritmus (6.15) je implementován v Příloze C2.

6.2.3 Variabilní délka kroku

Určení délky kroku v každé iteraci zvlášť je sice výpočetně náročnější, avšak otevírá nám to možnost minimalizace koercivní funkce. Uvažujme základní délku kroku h, směr minimalizace η a aktuální aproximaci x_k. Dále zvolme parametry $\mu \geq 1$, $0 < \gamma < 1$ a kladné celé číslo L. Místo výpočtu klasické aproximace však následuje výpočet pouze testovací aproximace

$$t_1 = x_k + h\eta. \tag{6.16}$$

Test spočívá ve vyhodnocení podmínky

$$f(t_1) < f(x_k). \tag{6.17}$$

V případě, že je podmínka (6.17) splněna, zkusíme další testovací aproximaci

$$t_2 = t_1 + h\eta. \tag{6.18}$$

Vyhodnotíme podmínku

$$f(t_2) < f(t_1). \tag{6.19}$$

Postup opakujeme, dokud podmínka platí, a ukončíme ve chvíli, kdy pro další testovací aproximaci platí

$$f(t_{K+1}) \geq f(t_K). \tag{6.20}$$

Pak pro skutečnou následující aproximaci platí

$$x_{k+1} = t_K. \tag{6.21}$$

Navíc pokud číslo K je větší než číslo L, provedeme následující úpravu základní délky kroku

$$K > L \implies h = \frac{K}{L} \cdot \mu h\,. \tag{6.22}$$

Význam procedury (6.22) je zřejmý – původní délka kroku byla příliš malá, proto jsme nastavili novou, větší délku. Navíc nikoli zkusmo, ale ze zkušenosti získané při hledání aktuální aproximace.

Může se ovšem stát, že podmínku (6.17) vyhodnotíme záporně hned v první testovací aproximaci (6.16). To znamená, že její funkční hodnota je vyšší než funkční hodnota současné aproximace. Pak platí

$$x_{k+1} = t_1\,. \tag{6.23}$$

A následně zmenšíme základní délku kroku

$$f(t_1) \geq f(x_k) \implies h = \gamma h\,. \tag{6.24}$$

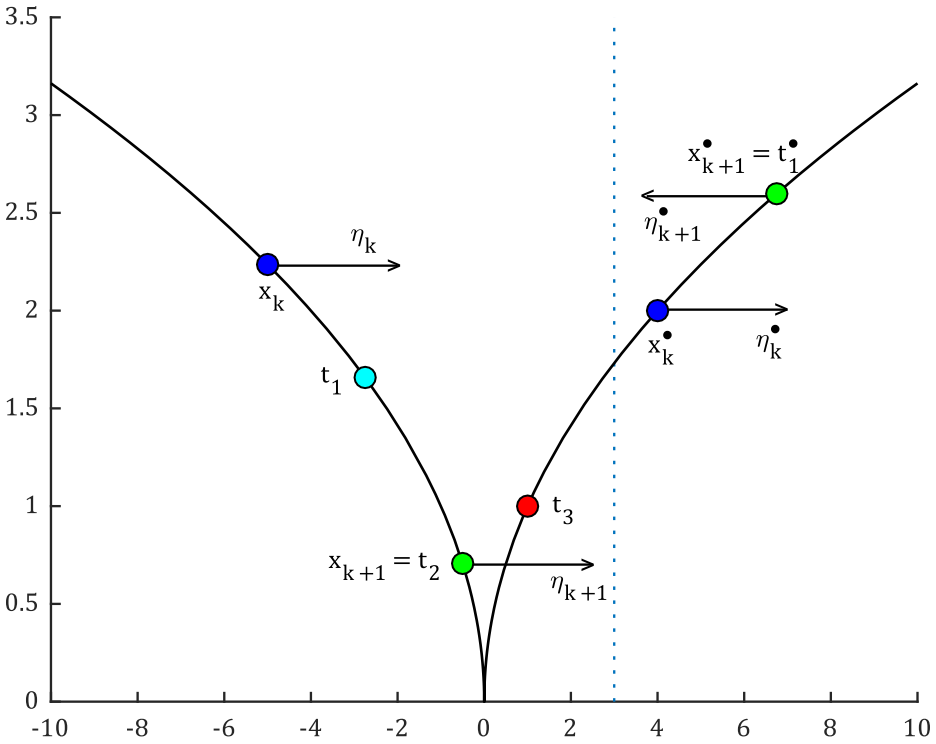
Pozor! Pořadí bodů (6.23) a (6.24) musíme dodržet. Nemáme totiž garantováno, že směr minimalizace η je skutečně směrem poklesu. Pokud by funkce ve směru η rostla a současně bychom nedodrželi dané pořadí, metoda by se zacyklila v neustálém snižování základní délky kroku. Nevýhodou tohoto postupu je, že nová aproximace má vyšší funkční hodnotou než současná.

V ostatních případech základní délku kroku neměníme. Tento výpočet nové aproximace vede ke snížení funkční hodnoty téměř v každé iteraci, jedinou výjimkou je situace popsaná výše.

Volbě parametrů h, μ a γ se budeme věnovat později, nyní si graficky znázorníme postup (6.16-6.24). Uvažujme funkci $f(x) = \sqrt{|x|}$, která je nehladká a konkávní, *Obrázek 13*. **Nový směr poklesu uvádíme pouze pro lepší pochopení principu**, ve skutečnosti je tento dán použitou metodou.

Předpokládejme, že v aproximaci x_k jsme zvolili skutečný směr poklesu. Nyní postupně testujeme funkční hodnoty v aproximacích t_1, t_2 a t_3. Ovšem $f(t_3) \geq f(t_2)$, proto nová aproximace řešení je $x_{k+1} = t_2$. Výchozí délka kroku v iteraci zůstane stejná, stejně jako uvažovaný směr poklesu.

Oproti tomu v aproximaci $x_k^\bullet$ jsme zvolili chybný směr poklesu. Dostaneme se do nové aproximace $x_{k+1}^\bullet = t_1^\bullet$, kde ale $f(t_1^\bullet) \geq f(x_k^\bullet)$. Proto dojde ke zmenšení výchozí délky kroku na $h = \gamma h$, a zvolíme nový směr poklesu $\eta_{k+1}^\bullet = -\eta_k^\bullet$.



Obrázek 13 – Rutina pro hledání délky kroku. Vlevo ukázka pro správně zvolený směr poklesu, vpravo ukázka pro chybně zvolený směr poklesu.

Algoritmus 6.25: ***Výpočet nové aproximace pro variabilní délku kroku***

i. **Vstup:** Funkce f, aktuální aproximace x_k, parametry h, γ, μ, L a směr poklesu η

ii. **Předpis algoritmu:**

$K \coloneqq 1$

$t_K \coloneqq x_k + h\eta$

if $f(t_K) \geq f(x_k)$

$x_{k+1} \coloneqq t_K$

$h \coloneqq \gamma h$

return x_{k+1}, h

end if

while $f(t_K) < f(x_{k+1})$

$x_{k+1} \coloneqq t_K$

$t_{K+1} \coloneqq t_K + h\eta$

$K \coloneqq K + 1$

end while

$K \coloneqq K - 1$

if $K > L$

$h = \frac{K}{L} \cdot \mu h$

end if

return x_{k+1}, h

Algoritmus (6.25) je implementován v Příloze C3. Směr poklesu η nemusí být jednotkový vektor.

6.2.4 Ukončovací podmínky výpočtu

Ukončit výpočet ve správnou chvíli je pro nehladkou funkci mnohem složitější než u hladké funkce. Nicméně i zde máme několik možností. Výpočet ukončíme, pokud některá z uvedených hodnot

1) Velikost subgradientu, ε_1 (**vždy**)
2) Velikost transformovaného subgradientu, ε_2 (pouze SDG a r-algoritmus, **zde vždy**)
3) Délka kroku, ε_3 (**vždy**)

4) Vzdálenost dvou po sobě jdoucích aproximací $\|x_{k+1} - x_k\|$, ε_4 (pouze SD metody)
5) Velikost rozdílu hodnot minima a aproximace minima $|f(\bar{x}) - f(\tilde{x})|$, ε_5

klesne pod určitou hodnotu, kterou označujeme jako přesnost výpočtu ε_i. Uvedené možnosti obvykle spojujeme logickou spojkou *nebo*. Pochopitelně, ne vždy máme k dispozici všechna uvedená kritéria.

Podmínky zastavení výpočtu musíme implementovat každou jednotlivě vždy na správném místě. Podmínky 1) a 2) mimo jiné zajišťují korektnost dělení (pokud má subgradient v dané iteraci nulovou velikost, výpočet se ukončí). Pozor, v rámci numerického výpočtu nulová velikost subgradientu ještě neznamená nalezení přesného řešení, protože se může jednat o zaokrouhlovací chybu.

Zavedeme **vektor přesnosti $\boldsymbol{\varepsilon}$**, do kterého zapíšeme požadované hodnoty. Pokud některé kritérium nelze nebo nechceme použít, jeho hodnotu nastavíme $\varepsilon_i = -1$. Ovšem mějme na paměti, že čím více ukončovacích kritérií, tím zpravidla lepší celkový výsledek.

$$\varepsilon = (\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5). \tag{6.26}$$

6.2.5 Metoda největšího spádu – NS

Algoritmus této metody vychází z klasické metody NS pro hladkou funkci. Subgradient $g_k \in \partial f(x_k)$ můžeme zvolit libovolný, numericky jej vypočteme algoritmem (5.28). Pro směr poklesu η platí

$$\eta_k = -\frac{g_k}{\|g_k\|}. \tag{6.27}$$

- V případě **konvexní cenové funkce** použijeme pro délku kroku h_k předpis (6.13), novou aproximaci získáme pomocí algoritmu (6.15). Pro neomezenou optimalizaci, $\Omega = \mathbb{R}^n$, můžeme použít postup pro koercivní funkce.

- V případě **koercivní cenové funkce** použijeme pro výpočet nové aproximace algoritmus (6.25).

Nyní můžeme uvést algoritmus metody NS. Ohodnocení všech potřebných parametrů ukážeme později na konkrétním příkladu.

Algoritmus 6.28: ***Metoda největšího spádu pro nehladkou funkci***

i. **Vstup:** Funkce f: $\mathbb{R}^n \to \mathbb{R}$, vektor přesnosti ε, volitelně hodnota minima $f(\bar{x})$

ii. **Inicializace:**
počáteční aproximace $x_0 \in \mathbb{R}^n$

$pokracuj \coloneqq 1$

$k \coloneqq 0$

* $c \in \mathbb{R} : c > \|\bar{x} - x_0\|$ *(pouze pro konvexní funkce)*

** základní délka kroku h *(pro konvexní i koercivní funkce)*

iii. Iterační cyklus

while $true$

$g_k \in \partial f(x_0)$

if $\|g_k\| < \varepsilon_1$

break

end if

$\eta_k \coloneqq -g_k / \|g_k\|$

*$x_{k+1} \coloneqq algoritmus$ 6.15 *(délka kroku daná předem)*

** $x_{k+1} \coloneqq algoritmus$ 6.25 *(variabilní délka kroku)*

$k \coloneqq k + 1$

if $DelkaKroku < \varepsilon_3$ ***or*** $\|x_{k+1} - x_k\| < \varepsilon_4$ ***or*** $|f(\bar{x}) - f(x_{k+1})| < \varepsilon_5$

break

end if

end while

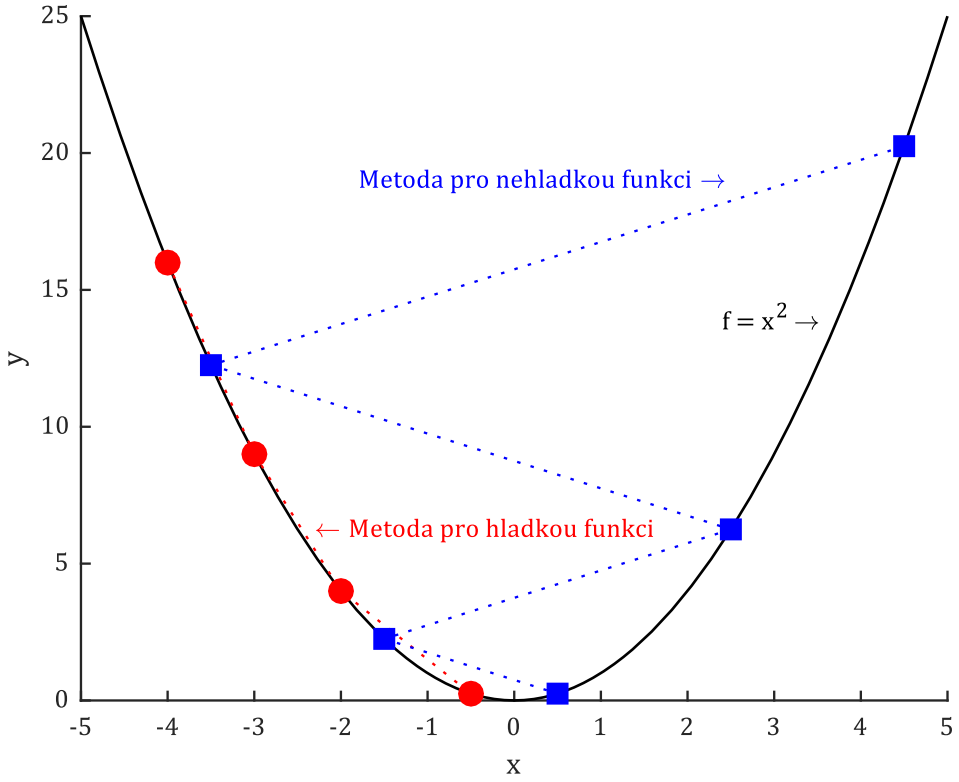
iv. Výstup:

$\tilde{x} \coloneqq x_k$

$iterace \coloneqq k$

Algoritmus je implementován v Příloze C4. Použijeme vždy jednu z možností označených * nebo **. Zdůrazněme, že ukončovací podmínku *„test velikosti subgradientu“,* nelze umístit až na konec iterace – hrozilo by dělení nulou. Ostatní ukončovací podmínky lze zařadit až na konec iterace.

Hlavní rozdíl v průběhu výpočtu metody největšího spádu pro hladkou funkci (s optimální délkou kroku) a nehladkou funkci (s délkou kroku danou předem) je dobře vidět na jednorozměrné úloze. Nechť je určen směr, ve kterém chceme minimalizovat, a přesné řešení úlohy. V případě metody pro hladkou funkci následující aproximace obvykle leží mezi aktuální aproximací a přesným řešením. Oproti tomu u metody pro nehladkou funkci se můžeme dostat i za přesné řešení (což je zcela běžné). Situaci znázorňuje *Obrázek 14* pro funkci $f(x) = x^2$.



Obrázek 14 – Srovnání průběhu minimalizace spojité funkce metodou největšího spádu pro hladkou a nehladkou funkci. Na obrázku jsou vyznačeny aproximace řešení v jednotlivých iteracích.

Rychlost konvergence je dána počtem iterací. V případě algoritmu (6.15) pro konvexní funkci můžeme vyjádřit časovou výpočetní náročnost v nejhorším případě jako počet iterací τ_{iter}. Předpokládejme přesnost kroku ε_3 a poloměr prohledávaného okolí c. Pak délku kroku v každé iteraci můžeme vyjádřit

$$h_k = \frac{c}{k+1}, \tag{6.29}$$

přičemž na konci každé iterace testujeme, zdali $h_k < \varepsilon_3$. V tom případě výpočet ukončíme. Pak

$$\tau_{iter} = k+1 \Leftrightarrow h_k = \frac{c}{k+1} < \varepsilon_3 \,. \tag{6.30}$$

Ze vztahu (6.30) snadno vyjádříme k v závislosti na c a přesnosti ε_3

$$\frac{c}{k+1} < \varepsilon_3 \implies c < \varepsilon_3 k + \varepsilon_3 \implies k > \frac{c-\varepsilon_3}{\varepsilon_3} \implies k = \left\lceil \frac{c}{\varepsilon_3} \right\rceil - 1 \,.$$

Nyní již můžeme vyjádřit počet iterací v nejhorším případě

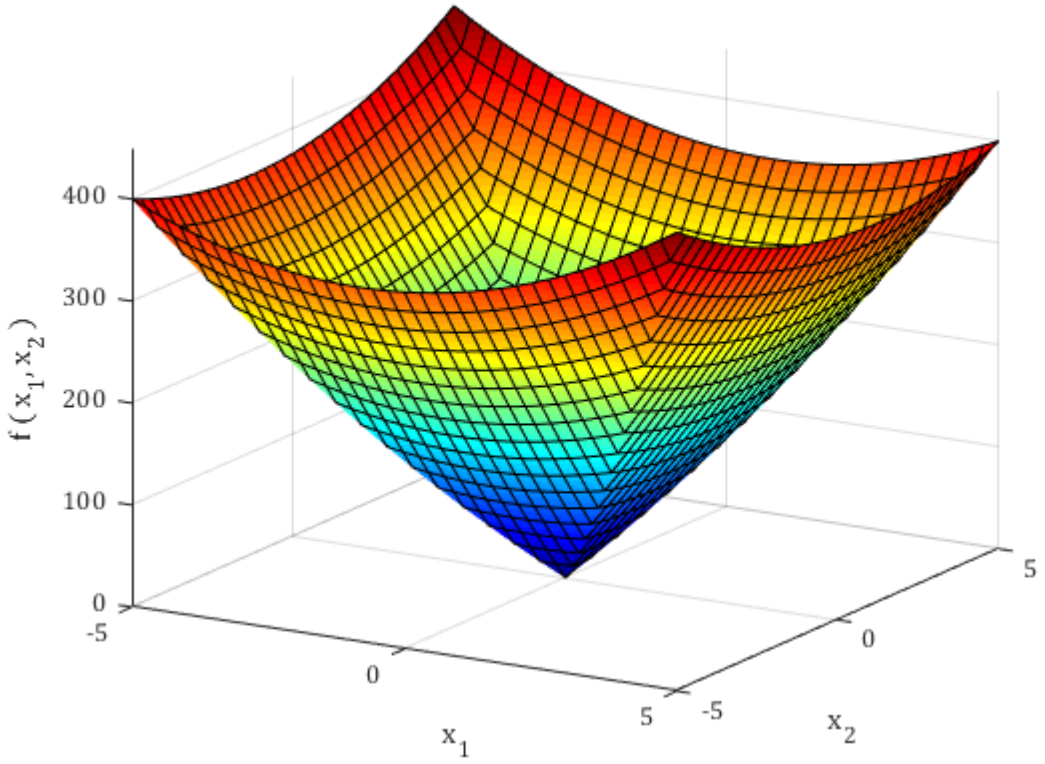
$$\tau_{iter} = k+1 = \left\lceil \frac{c}{\varepsilon_3} \right\rceil. \tag{6.31}$$

Výpočetní náročnost je v tomto případě známá dopředu, a lze ji plně nastavit vhodnou volbou parametrů c a ε_3, což je výhodné pro výpočty v reálném čase. Efektivitu metody NS s výpočtem nové aproximace algoritmem (6.15) zásadně ovlivňuje skutečnost, s jakou přesností umíme odhadnout přibližné řešení.

Pro algoritmus (6.25) počet iterací dopředu stanovit neumíme. Pokud potřebujeme zajistit maximální délku výpočtu předem, můžeme stanovit maximální přípustný počet iterací nebo časový limit výpočtu (jako další ukončovací kritéria), případně celý problém svěřit nadřazenému programu (SW řešení) nebo obvodu (HW řešení), které výpočet ukončí.

Příklad 6.32: Optimalizujte danou cenovou funkci. Použijte všechny dostupné metody, předpokládejte neznalost hodnoty minima. Analyzujte získané výsledky. Graficky znázorněte průběh výpočtu.

$$f(x_1, x_2) = 3x_1^2 + 2x_2^2 - x_1x_2 + 30|x_1 + x_2| + 30|x_1 - x_2|\,.$$



Obrázek 15 – K příkladu 6.32, graf koercivní funkce.

Řešení: Nejdříve musíme ověřit konvexitu a koercivitu funkce. Konvexitu dokážeme následovně. Pokud jsou všechny sčítance konvexní funkce, pak i jejich součet je konvexní funkce. Funkci přepíšeme na tvar

$$f(x_1, x_2) = (3x_1^2 + 2x_2^2 - x_1x_2) + 30(|x_1 + x_2| + |x_1 - x_2|).$$

Pro sčítanec $(3x_1^2 + 2x_2^2 - x_1x_2)$ vypočteme Hessián

$$H^* = \begin{pmatrix} 6 & -1 \\ -1 & 4 \end{pmatrix}, \det H_1^* = 6, \det H_2^* = 23.$$

Hessián je pozitivně definitní, tudíž sčítanec $(3x_1^2 + 2x_2^2 - x_1x_2)$ je dokonce ryze konvexní funkce. V případě druhého sčítance, $30(|x_1 + x_2| + |x_1 - x_2|)$, nelze Hessián použít, musíme vyjít z definice. Tím se ovšem nebudeme zabývat. Chtěli jsme pouze naznačit, že důkaz splnění předpokladů nemusí být vůbec triviální. Druhý sčítanec je neryze konvexní funkce, avšak s ostrým globálním minimem. Cenová funkce je tedy konvexní a globální minimum existuje jediné.

Pokusíme se odhadnout přibližné řešení úlohy. Z *Obrázku 15* je patrné, že řešení $\bar{x} \in \langle -5; 5\rangle \times \langle -5; 5\rangle$. Pro ukázku výpočtu zvolíme tři typově odlišné počáteční aproximace, všechny parametry jsou uvedeny v následující tabulce.

Test	x_0	Délka kroku daná předem, alg. (6.15)		Variabilní délka kroku, algoritmus (6.25)				
		c	ε	h	γ	μ	L	ε
1	$(0;\ 0)^T$	7,5	ε^1	0,1	0,5	2	5	ε^3
2	$(5;\ 5)^T$	15	ε^1	0,1	0,5	2	5	ε^3
3	$(10^6; 10^6)^T$	$3{\cdot}10^6$	ε^2	100	0,5	2	5	ε^4

Tabulka 9 – Nastavení počátečních hodnot pro minimalizaci.

Výpočet ideální hodnoty parametru c. Víme, že $\bar{x} \in M$, kde $M = \langle -5; 5\rangle \times \langle -5; 5\rangle$, proto

$$c > \sup\{\rho(x_0, y) \mid y \in M = \langle -5; 5\rangle \times \langle -5; 5\rangle\}.$$

Parametr c není potřeba počítat přesně, stačí jeho spodní odhad. Pro libovolnou počáteční aproximaci chceme určit největší vzdálenost od libovolného bodu množiny M. Odhadneme

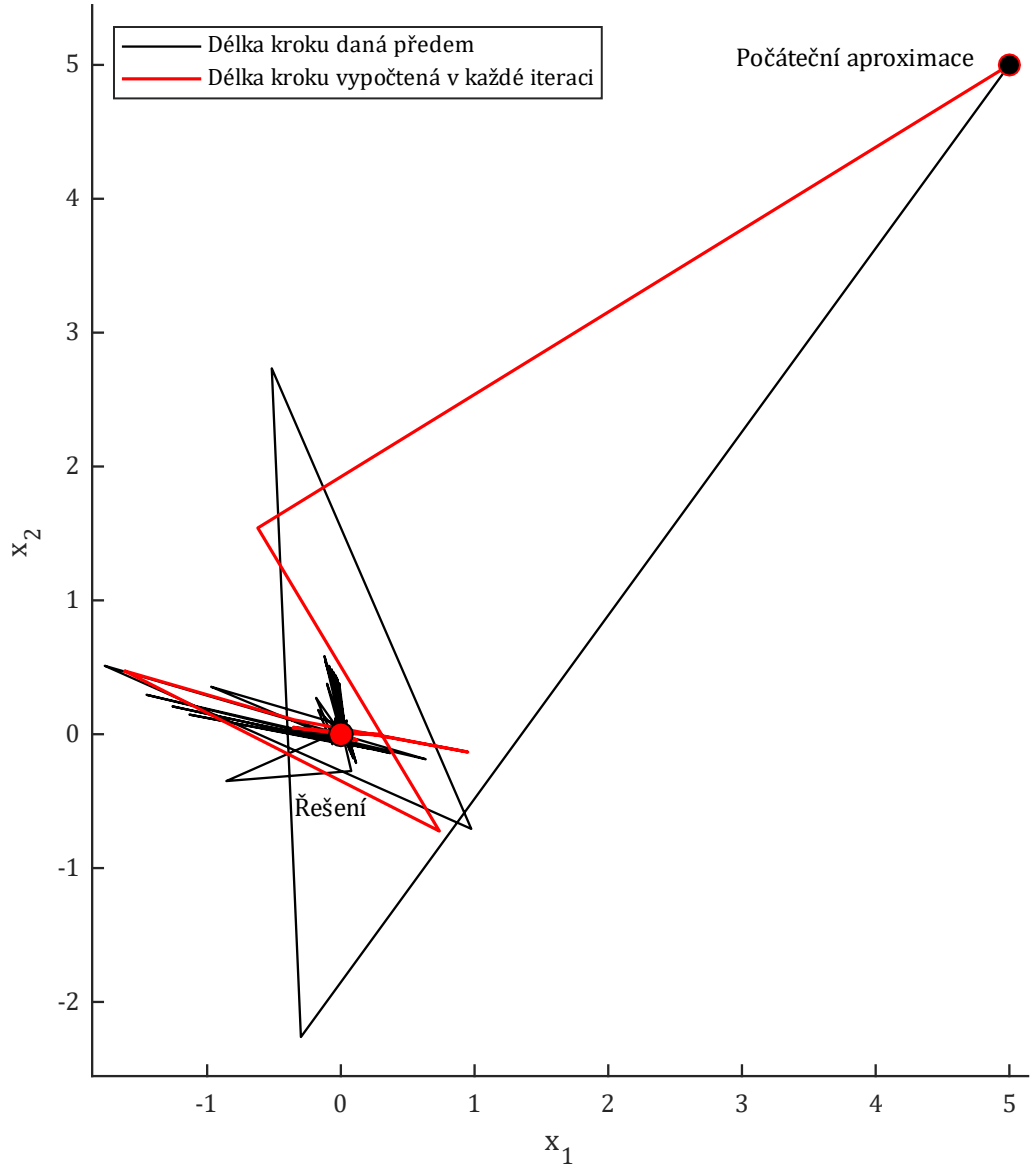
$$x_0^1 = (0;0)^T \implies c_1 > 5\sqrt{2} \implies c_1 \overset{\text{def}}{=} 7{,}5\,,$$
$$x_0^2 = (5;5)^T \implies c_2 > 10\sqrt{2} \implies c_2 \overset{\text{def}}{=} 15\,,$$
$$x_0^3 = (10^6;10^6)^T \implies c_3 > 10^6\sqrt{8} \implies c_3 \overset{\text{def}}{=} 3 \cdot 10^6.$$

Bohužel, pro variabilní délku kroku neexistuje žádné exaktní kritérium pro stanovení hodnot parametrů, musíme vycházet z naší zkušenosti a citu. Můžeme zvolit *„agresivnější nastavení“* s vyššími hodnotami parametrů h a μ a nižšími hodnotami γ a L. Nebo naopak *„defenzivní nastavení“*. Výhodou *„agresivního nastavení“* je vyšší rychlost výpočtu, nevýhodou pak jeho menší stabilita.

Stanovení přesnosti provedeme podle přesnosti odhadu počáteční aproximace.

	Algoritmus (6.15)	Algoritmus (6.25)
$\varepsilon^1 = (10^{-4}; -1; 10^{-4}; -1; -1)$	pro x_0^1 a x_0^2	pro x_0^1, x_0^2 i x_0^2
$\varepsilon^2 = (10^0; -1; 10^0; -1; -1)$	pro x_0^3	---

Tabulka 10 – Nastavení přesnosti.



Obrázek 16 – Srovnání průběhu výpočtu pro počáteční aproximaci $x_0 = (5,5)^T$.
Černě délka kroku daná předem, červeně variabilní délka kroku.

Řešení je implementováno v Příloze D1. Zde si ukážeme pouze dosažené výsledky – *Tabulka 11*. Přesné analytické řešení je $\bar{x} = (0; 0)^T$ a $f(\bar{x}) = 0$ (stačí použít výše zmíněný rozklad na součet dvou funkcí). Nakonec zobrazíme průběh výpočtu u obou variant – vykreslíme posloupnost postupných aproximací řešení pro menší přesnost – *Obrázek 16*.

x_0	Algoritmus (6.15)					Algoritmus (6.25)				
	$\Delta\tilde{x}$	$\Delta f(\tilde{x})$	Kód	Iterace	Čas	$\Delta\tilde{x}$	$\Delta f(\tilde{x})$	Kód	Iterace	Čas
$\binom{0}{0}$	0	0	1	0	56 μs	0	0	1	0	75 μs
$\binom{5}{5}$	$4{,}4 \cdot 10^{-9}$	$2{,}4 \cdot 10^{-7}$	1	$2 \cdot 10^{4}$	0,13 s	$1{,}2 \cdot 10^{-4}$	$7{,}4 \cdot 10^{-3}$	3	41	0,54 ms
$\binom{10^6}{10^6}$	0,90	56	3	$3 \cdot 10^{6}$	19 s	$9{,}9 \cdot 10^{-5}$	$4{,}3 \cdot 10^{-3}$	3	68	5,4 ms

Tabulka 11 – Výsledky úlohy a výpočetní náročnost jednotlivých metod. Kód = typ ukončovací podmínky podle kapitoly 6.6.Podbarvení ukazuje zásadní rozdíly ve výpočetní náročnosti.

Závěr: Je zřejmé, že **použití variabilní délky kroku je mnohem efektivnější než délka kroku daná předem**. Jednotlivé iterace jsou sice výpočetně náročnější, avšak celkový počet iterací je o několik řádů nižší. Navíc variabilní délka kroku nám umožňuje minimalizaci širšího spektra funkcí.

△

Důsledek 6.33: Při optimalizaci raději volíme variabilní délku kroku vypočtenou v každé iteraci. V případě omezené optimalizace lze podmínku koercivity splnit vhodným dodefinováním cenové funkce na množině $\mathbb{R}^n \setminus \Omega$, kde Ω je množina přípustných řešení.

6.2.6 Minimalizace s dilatací prostoru – metoda SDG

Ukážeme si jiný přístup k řešení, podobný metodě sdružených gradientů. Začněme vysvětlením pojmů. **SDG** znamená *„Space Dilation Along the Gradient“*, česky „*Subgradientní metoda s dilatací prostoru ve směru gradientu“*. U nehladké funkce použijeme Clarkeův zobecněný gradient (resp. subgradient).

SDG metoda kombinuje minimalizaci funkce ve směru subgradientu s dilatací prostoru v tomto směru. Budeme tak řešit **tři podúlohy**

- **Výpočet subgradientu** $g_k \in \partial f(x_k)$,
- **Dilataci prostoru,**
- **Délku kroku** h_k.

Výpočet subgradientu $g_k \in \partial f(x_k)$ již známe, zde se nic nemění. Co se délky kroku týče, použijeme pouze algoritmus (6.25), tedy stanovíme optimální délku kroku v každé iteraci.

Pro dilataci prostoru potřebujeme určit vektor g_k^*, jehož normovaný tvar ξ_k použijeme jako argument operátoru $R_\alpha(\xi_k)$ a také matici transformace A. Nejdříve uvažujme transformaci k-té aproximace x_k, kterou postupně upravíme.

$$y_k = A_k x_k \ \ | \cdot A_k^{-1}\ zleva, \tag{6.34}$$

$$A_k^{-1} y_k = A_k^{-1} A_k x_k \implies x_k = A_k^{-1} y_k. \tag{6.35}$$

Získali jsme k-tou aproximaci jako funkci φ_k proměnné y

$$x_k = \varphi_k(y), \tag{6.36}$$

$$\varphi_k(y) = f\left(A_k^{-1} y\right). \tag{6.37}$$

Nyní zavedeme matici B_k inverzní k A_k a dosazením do vztahu (6.37) získáme (6.39)

$$B_k = A_k^{-1} \iff A_k = B_k^{-1}, \tag{6.38}$$

$$\varphi_k(y) = f(B_k y). \tag{6.39}$$

Hledaný vektor g_k^* je subgradientem pomocné funkce φ_k. Pomocí matice B_k^* jej můžeme zapsat

$$g_k^* = B_k^* g_k. \tag{6.40}$$

V angličtině mají pro matici B_k^* termín *„adjoint operator“*. V rámci $\mathbb{R}^n$ pro *„adjoint operator“* platí

$$g_k^* = B_k^* g_k \iff (B_k y, g_k) = (y, g_k^*) \quad \implies \quad (B_k y, g_k) = (y, B_k^* g_k), \tag{6.41}$$

$$(B_k y)^T g_k = y^T B_k^* g_k \implies y^T B_k^T g_k = y^T B_k^* g_k \implies y^T \left(B_k^T - B_k^*\right) g_k = o. \tag{6.42}$$

Ze vztahu (6.42) již snadno získáme předpis pro matici B_k^* bez ohledu na vektory $y = Ax$ a g. Vztah (6.45) dále zavádí normovaný tvar g_k^*.

$$B_k^* = B_k^T, \tag{6.43}$$

$$g_k^* = B_k^T g_k, \tag{6.44}$$

$$\xi_k = \frac{g_k^*}{\left\| g_k^* \right\|}. \tag{6.45}$$

Vektor ξ_k použijeme jako směr dilatace. Matice transformace má pak následující rekurzivní předpis

$$A_{k+1} = R_\alpha(\xi_k) A_k. \tag{6.46}$$

Pro matici B_{k+1} podle (6.38) platí

$$B_{k+1} = A_{k+1}^{-1} = (R_\alpha(\xi_k)A_k)^{-1} = A_k^{-1}R_\alpha^{-1}(\xi_k). \tag{6.47}$$

Ale $A_k^{-1} = B_k$ a pro matici dilatace platí $R_\alpha^{-1}(\xi_k) = R_{1/\alpha}(\xi_k)$. Dosazením do vztahu (6.47) získáme rekurzivní předpis pro výpočet matice B, který dále upravíme. Vše plyne z věty (6.9).

$$B_{k+1} = B_k R_{1/\alpha}(\xi_k) = B_k\big(I + (1/\alpha\ - 1)\xi_k\xi_k^T\big). \tag{6.48}$$

Nyní již víme, jak spočítat subgradient g_k^* i matici B_k. Matici A k výpočtu potřebovat nebudeme. Za výchozí matici A_0 obvykle volíme jednotkovou matici, pak i matice B_0 je jednotková. Vlastnosti matice B můžeme dále vylepšit následující procedurou. Po každých p iteracích zjistíme největší velikost prvku matice B (velikostí myslíme absolutní hodnotu)

$$d = \max_{i,j}|b_{ij}|. \tag{6.49}$$

Pokud $d < \delta \in \mathbb{R}^+$, pak matici B vynásobíme koeficientem $\rho > 1$. Tuto rutinu provádíme opakovaně, dokud není splněna podmínka $d < \delta$. Parametry δ a ρ jsou konstanty. Lepší způsob výpočtu představuje výpočet koeficientu pro násobení matice B předem.

$$d_s = d\rho^s\ \wedge\ d_s \geq \delta\ \Longrightarrow\ d\rho^s \geq \delta\ \Longrightarrow\ \rho^s \geq \frac{\delta}{d}, \tag{6.50}$$

označme

$$\vartheta \overset{\text{def}}{=} \frac{\delta}{d}. \tag{6.51}$$

Nyní v každé p-té iteraci, pokud $\vartheta > 1$, vynásobíme matici B koeficienty ϑ a $\nu > 1$

$$\vartheta > 1\ \Longrightarrow\ B = \nu\vartheta\mathrm{B}. \tag{6.52}$$

Hodnoty parametrů p, δ a ν objasníme později. Smysl této operace je zřejmý – velikost prvků matice B se během výpočtu může zmenšovat, čímž dochází k většímu vlivu zaokrouhlovacích chyb. Uvedenou rutinou tomu snadno předejdeme. Uvedeme ještě její algoritmus.

Algoritmus 6.53: ***Optimalizace matice B***

i. **Vstup:** Matice B, parametry δ a ν

ii. **Optimalizace:**

$$d \coloneqq \max_{i,j}|b_{ij}|$$

$\vartheta \coloneqq \delta / d$

if $\vartheta > 1$

$B \coloneqq \nu\vartheta \mathrm{B}$

end if

iii. Výstup:

B

Implementaci uvádíme v Příloze C5.

Poslední neznámou je koeficient dilatace prostoru $\alpha_k \geq 1$, který sice není přesně určen, avšak současně má velký vliv na výpočet. Můžeme jej zvolit konstantní, $\alpha_k = \alpha$, ale stejně tak jej můžeme měnit v každé iteraci. Pro $B_0 = I$ a $\alpha = 1$ získáme metodu největšího spádu z minulé kapitoly. Vhodné volbě koeficientu se budeme věnovat později (jsou částečně závislá na dalších faktorech).

Algoritmus 6.54: ***SDG metoda***

i. Vstup: Funkce $f\colon \mathbb{R}^n \to \mathbb{R}$, vektor přesnosti ε, koeficient dilatace prostoru α, volitelně hodnota minima $f(\bar{x})$

ii. Inicializace:

počáteční aproximace $x_0 \in \mathbb{R}^n$

$pokracuj \coloneqq 1$

$B_0 \coloneqq I$

$k \coloneqq 0$

$\kappa \coloneqq 1/\alpha \ - 1$

základní délka kroku h

iii. Iterační cyklus

while $true$

$g_k \in \partial f(x_0)$

if $\|g_k\| < \varepsilon_1$

break

end if

$g_k^* \coloneqq B_k^T g_k$

if $\|g_k^*\| < \varepsilon_2$

break

end if

$\xi_k \coloneqq g_k^*/\|g_k^*\|$

$\eta_k \coloneqq -B_k\xi_k$

$x_{k+1} \coloneqq algoritmus\ 6.25$

$B_{k+1} \coloneqq B_k \cdot (I + \kappa \cdot \xi_k\xi_k^T)$

$Optimalizuj\ matici\ B_{k+1} - algoritmus\ 6.53$

$k \coloneqq k + 1$

if $DelkaKroku < \varepsilon_3$ ***or*** $\|x_{k+1} - x_k\| < \varepsilon_4$ ***or*** $|f(\bar{x}) - f(x_{k+1})| < \varepsilon_5$

break

end if

end while

iv. Výstup:

$\tilde{x} \coloneqq x_k$

$iterace \coloneqq k$

Implementaci uvádíme v Příloze C6. Výhodou SDG metody je použití dvou řad koeficientů – $\{h_k\}$ a $\{\alpha_k\}$, což umožňuje lepší optimalizaci výpočtu pro různé typy úloh (širší spektrum úloh).

Nemusíme provádět žádné složité experimenty, abychom zjistili, že výpočetní náročnost SDG metody při stejném počtu iterací je vyšší než metody největšího spádu. Náročný je zejména výpočet vektoru g_k^*, nové aproximace x_{k+1} a matice B_{k+1}, každý z nich vyžaduje přibližně $4n^2$ násobení. Tato skutečnost nás vede k následujícímu velmi důležitému závěru.

Důsledek 6.55: Způsob výpočtu délky kroku

Pro metody s dilatací prostoru (SDG metodu nevyjímaje) je žádoucí použití variabilní délky kroku. Použití délky kroku dané předem je výpočetně výrazně náročnější, a dokonce můžeme získat horší výsledky než při použití metody největšího spádu.

Příklad 6.56: Minimalizujte funkci z příkladu (6.32) metodou SDG. Použijte stejné hodnoty parametrů včetně počátečních aproximací. Získané výsledky porovnejte, použijte variabilní délku kroku. Výpočet dále proveďte pro řádově vyšší přesnost.

Řešení: Vše již máme připravené, pouze si ukážeme veškerá nastavení

ε^1	$(10^{-4}; 10^{-8}; 10^{-6}; 10^{-5}; -1)$
ε^2	$(10^{-8}; 10^{-16}; 10^{-9}; 10^{-9}; -1)$
ε^3	$(10^{-12}; 10^{-24}; 10^{-14}; 10^{-14}; -1)$

Tabulka 12 – Nastavení přesnosti, k příkladu 6.56.

	x_0	**Délka kroku**				**Dilatace prostoru**			
		h	γ	μ	L	α	p	δ	ν
1	$(0;\ 0)^T$	0,1	0,1	1,25	5	2,0	10	1	10
2	$(5;\ 5)^T$	0,1	0,1	1,25	5	2,0	10	1	10
3	$(10^6; 10^6)^T$	100	0,1	1,25	5	1,3	10	1	10

Tabulka 13 – Nastavení počátečních hodnot pro minimalizaci, k příkladu 6.56.

Řešení je implementováno v Příloze D2. Přesné analytické řešení je $\bar{x} = (0; 0)^T$ a $f(\bar{x}) = 0$. Naměřené hodnoty pro přesnost ε^1 jsme zaznamenali do *Tabulky 14* (včetně srovnání s metodou NS).

x_0	**Metoda NS, přesnost ε^1**					**SDG metoda, přesnost ε^1**				
	$\Delta\tilde{x}$	$\Delta f(\tilde{x})$	Kód	Iterace	Čas	$\Delta\tilde{x}$	$\Delta f(\tilde{x})$	Kód	Iterace	Čas
$\binom{0}{0}$	0	0	1	0	56 µs	0	0	1	0	70 µs
$\binom{5}{5}$	4,4 ·10^{-9}	2,4 ·10^{-7}	1	2·10^4	0,13 s	3,8 ·10^{-6}	1,8 ·10^{-4}	4	37	0,68 ms
$\binom{10^6}{10^6}$	0,90	56	3	3·10^6	19 s	2,7 ·10^{-6}	1,4 ·10^{-4}	4	48	5,1 ms

Tabulka 14 – Srovnání výpočetní náročnosti jednotlivých metod. Kód = typ ukončovací podmínky podle kapitoly. Podbarvení značí rozdíly v přesnosti získaného řešení.

Z tabulky je patrné, že při *zhruba stejném* nastavení přesnosti jsme získali řešení s řádově menší chybou, a to při stejné výpočetní náročnosti. Výsledky pro zbylé přesnosti ε^2 a ε^3 uvádíme v *Tabulce 15*.

x_0	Přesnost ε^2					Přesnost ε^3				
	$\Delta\tilde{x}$	$\Delta f(\tilde{x})$	Kód	Iterace	Čas	$\Delta\tilde{x}$	$\Delta f(\tilde{x})$	Kód	Iterace	Čas
$\binom{5}{5}$	5,1 $\cdot 10^{-8}$	2,4 $\cdot 10^{-6}$	3	61	1,0 ms	1,3 $\cdot 10^{-13}$	6,9 $\cdot 10^{-12}$	3	101	1,5 ms
$\binom{10^6}{10^6}$	2,8 $\cdot 10^{-9}$	1,5 $\cdot 10^{-7}$	3	65	5,2 ms	1,1 $\cdot 10^{-13}$	6,7 $\cdot 10^{-12}$	3	82	5,7 ms

Tabulka 15 – Řešení úlohy 6.56.

Zdá se, že řešení minimalizační úlohy můžeme získat v reálném čase prakticky pro libovolnou zadanou přesnost. Znovu ale připomeňme, že přesnost výpočtu uvažujeme ve smyslu určitého kritéria, primárně zavedeného pro zastavení výpočtu. Zde ale vzniká problém – takto zavedená přesnost ještě nutně nemusí znamenat skutečnou přesnost řešení – což ukážeme v numerických experimentech.

△

Na závěr kapitoly dodejme, že **hlavní výhoda metody SDG spočívá v dalších řadách nastavitelných koeficientů, díky kterým můžeme její použití lépe optimalizovat pro různé třídy funkcí.**

6.2.7 Minimalizace se znalostí funkční hodnoty v minimu

Uvažujme situaci, kdy sice neznáme polohu minima cenové funkce, ale známe funkční hodnotu v tomto minimu. To nám při minimalizaci může významně pomoci, zejména pak v úlohách ekonomického typu, ve kterých chceme primárně najít co nejnižší funkční hodnotu. Rovněž můžeme implementovat další ukončovací kritérium výpočtu. Protože jsme s touto variantou počítali již při implementaci algoritmu, můžeme rovnou přejít k ukázkové úloze.

Příklad 6.57: Minimalizujte funkci z příkladu (6.32) se znalostí funkční hodnoty v minimu $f(\bar{x}) = 0$. Počáteční aproximaci zvolte $x_0 = (5; 5)^T$. Požadovaná přesnost řešení je $\varepsilon_5 = |f(\bar{x}) - f(\tilde{x})| = 10^{-4}$.

Řešení: Znalost funkční hodnoty v minimu využijeme jako primární ukončovací podmínku. Vektor přesnosti výpočtu stanovíme $\varepsilon^1 = (10^{-20}; -1; -1; -1; 10^{-4})$ pro metodu největšího spádu s délkou kroku danou předem, ve všech ostatních případech použijeme $\varepsilon^2 = (10^{-20}; 10^{-20}; -1; -1;\ 10^{-4})$.

Poslední souřadnice vektoru představuje právě maximální chybu ε_5, tedy odchylku funkční hodnoty. První, respektive první dvě souřadnice vektoru jsou **normy subgradientů, které pro následující iteraci výpočtu nesmí být nulové.** Pokud $\|g_k\| = 0$, respektive $\|g_k^*\| = 0$, pak aktuální iterace x_k představuje minimum a výpočet končí. Primárně nám jde ovšem o to, abychom v algoritmu nedělili nulou, jelikož kritérium $|f(\bar{x}) - f(\tilde{x})|$ testujeme až jako poslední. Řešení je implementováno v Příloze D3.

Nabízí se otázka, zdali v takto koncipované úloze získáme řešení se skutečnou přesností, co se odchylky funkční hodnoty týče. Odpověď zní, bohužel, nikoli. Dopředu totiž nevíme, která z podmínek výpočet ukončí. K tomu může dojít i na základě dosažení přesnosti (velikosti) normy subgradientů. Tento případ je v praxi navíc zcela běžný, jak uvidíme v numerických experimentech. Samozřejmě, pokud ověříme, že výpočet ukončila podmínka pro maximální odchylku funkční hodnoty ε_5, pak můžeme deklarovat skutečnou přesnost aproximace řešení.

	Délka kroku					**Dilatace prostoru**			
	c	h	γ	μ	L	α	p	δ	ν
NS – krok předem	15								
NS – variabilní krok		0,1	0,1	1,25	5				
SDG – variabilní krok		0,1	0,1	1,25	5	2,2	10	1	10

Tabulka 16 – Nastavení počátečních hodnot pro minimalizaci.

NS – krok předem			**NS – variabilní krok**			**SDG – variabilní krok**		
Iterace	Čas	Δx	Iterace	Čas	Δx	Iterace	Čas	Δx
8,3 $\cdot 10^{6}$	60 s	4,3 $\cdot 10^{-10}$	21	0,53 ms	8,3 $\cdot 10^{-7}$	37	0,87 ms	1,7 $\cdot 10^{-5}$

Tabulka 17 – Srovnání výpočetní náročnosti jednotlivých metod při znalosti hodnoty v minimu. Jediným ukončovacím kritériem je přesnost funkční hodnoty v bodě minima.

Výpočetní náročnost pro metodu největšího spádu s délkou kroku danou předem je opravdu vysoká. Metoda SDG je v tomto případě o něco pomalejší a méně přesná než metoda NS s variabilní délkou kroku. Ovšem nevyvozujme z toho obecný závěr, že při znalosti funkční hodnoty v minimu je metoda NS s variabilní délkou kroku rychlejší (stačí zvolit jinou požadovanou přesnost a výsledek je opačný). Podstatné na tomto příkladu bylo ukázat, že **znalost funkční hodnoty v minimu nám může sloužit jako další ukončovací kritérium**.

△

6.2.8 Souhrn poznatků a konvergence metod

Z našeho pozorování můžeme vyslovit následující závěry (ve formě hypotéz)

- Z hlediska výpočetní náročnosti je **zcela zásadní určení délky kroku v každé iteraci zvlášť**.
- Znalost hodnoty minima nám pomáhá optimalizovat výpočet, zejména řeší otázku jeho ukončení.
- SDG metodu lze velmi dobře použít, i když neznáme odhad řešení. To je velice užitečná vlastnost, pokud minimalizujeme *neznámou* funkci. Nemusíme se tak nutně zabývat odhadem řešení.
- Určitou nevýhodou SDG metody může být nutnost vhodně nastavit vstupní parametry. To vyžaduje zkušenost a cit v dané problematice, případně použití osvědčených hodnot. Nastavením parametrů se budeme zabývat ještě v následujících kapitolách.

Samostatnou kapitolou je důkaz korektnosti implementovaných řešení. Obvykle se provádí pouze důkaz konvergence iteračního předpisu, přičemž se uvažuje přesný analytický model. Tyto důkazy můžeme najít v literaturách [1], [5], [6], [9] a dalších. V případě metod s variabilní délkou kroku nebo metod s dilatací prostoru se při důkazu často uvažuje jejich zjednodušení. Není výjimkou, že nakonec vlastně dokazujeme konvergenci postupu velice podobného metodě největšího spádu. Důkaz konvergence metody NS je základním stavebním kamenem pro důkazy ostatních metod.

Numerický model se vlivem zaokrouhlovacích chyb může chovat odlišně, v krajním případě můžeme získat nesmyslné řešení. Na to musíme brát zřetel, a zejména u výpočtů s vysokým počtem iterací jednou za čas provést tzv. regularizaci některých parametrů, typicky matic (při požadavku na jejich regularitu). To se provádí různými postupy, ten nejjednodušší spočívá v tzv. resetování matic, tedy jejich nastavení na původní hodnotu, obvykle na jednotkovou matici. My jsme v SD metodách použili algoritmus (6.60), který představuje sofistikovanější způsob regularizaci. Touto problematikou se detailně zabývá kniha *Počítačová algebra* [4].

Citlivost a potenciální nestabilita se u jednotlivých metod výrazně liší. Obecně však platí, že metody využívající více řad nastavovacích parametrů jsou stabilnější, protože nevýhody plynoucí z určitého nastavení jednoho parametru lze korigovat nastavením ostatních parametrů. Takové metody jsou tudíž vhodné pro optimalizaci mnohem širšího spektra funkcí.

7 Shorův r-algoritmus

Problematikou minimalizace koercivní funkce se, mimo jiné, zabýval N. Z. Shor. Výsledkem jeho práce je třída algoritmů souhrnně nazývaných r-algoritmy. Při jejich tvorbě navázal na práci N. G. Zhurbenka a L. P. Shabashové.

7.1 Naum Zuselevich Shor

Obrázek 17 – Naum Z. Shor, Наум Зуселевич Шор.

Byl ukrajinský i sovětský matematik (měl obě národnosti), narozený 1. ledna 1937 v Kyjevě na Ukrajině. Ve vědecké činnosti se zaměřil zejména na subgradientní metody s dilatací prostoru, jejich vývoji se věnoval více než patnáct let své kariéry. Za svou práci získal řadu ocenění, včetně těch nejvyšších (Ukrajinské státní vyznamenání za vědu a technologii, Sovětské státní vyznamenání). V roce 1998 se stal plnohodnotným členem ukrajinské Akademie věd, kde působil v Institutu kybernetiky. Dožil se 69 let, zemřel 26. února 2006.

Poznámka: Ukrajina byla až do roku 1991 součástí Sovětského svazu (USSR), byť už v roce 1918 se prohlásila za nezávislé území.

7.2 Počítač BESM-6

Obrázek 18 - Počítač BESM-6.

Počítač vystavený v Londýnském přírodovědeckém muzeu. Jedná se o první ruský tranzistorový počítač. Vyráběl se v letech 1968-87, jeho výkon byl při 9 MHz jeden milion instrukcí za sekundu. Paměť měla velikost 192 kB. Na tomto počítači prováděl výpočty právě N. Z. Shor.

Zdroj: Wikipedia (obrázky i text)

7.3 Základní varianta r-algoritmu

Hlavní myšlenkou tohoto algoritmu je interpolace řešení získaných pomocí metod sdružených gradientů a největšího spádu. Dilatace prostoru uvažujeme ve směru určeném rozdílem dvou po sobě následujících subgradientů. Dá se říct, že tato metoda kombinuje vlastnosti všech dříve uvedených metod. Ke snížení funkční hodnoty (minimalizaci) dochází v každé nebo téměř každé iteraci.

Vše potřebné jsme již probrali dříve, rovnou tedy uvedeme r-algoritmus. Použijeme výhradně variabilní délku kroku. Ukončovací podmínky výpočtu jsme stanovili v kapitole 6.2.4.

Algoritmus 7.1: ***Základní varianta r-algoritmu***

i. **Vstup:** Funkce $f: \mathbb{R}^n \to \mathbb{R}$, vektor přesnosti ε, koeficient dilatace prostoru α, volitelně hodnota minima $f(\bar{x})$

ii. **Inicializace:**

počáteční aproximace $x_0 \in \mathbb{R}^n$

$pokracuj \coloneqq 1$

$B_0 \coloneqq I$

$k \coloneqq 0$

$\kappa \coloneqq 1/\alpha \ - 1$

základní délka kroku h

$g_0 \in \partial f(x_0)$

$\tilde{g}_0 \coloneqq g_0$

iii. **Iterační cyklus**

while $true$

$g_k \in \partial f(x_k)$

if $\|g_k\| < \varepsilon_1$

break

end if

$g_k^* \coloneqq B_k^T g_k$

if $\|g_k^*\| < \varepsilon_2$

break

end if

$r_k \coloneqq g_k^* - \tilde{g}_k$

$\xi_{k+1} \coloneqq r_k / \|r_k\|$

$B_{k+1} \coloneqq B_k(I + \kappa \cdot \xi_{k+1}\xi_{k+1}^T)$

$Optimalizuj\ matici\ B_{k+1} - algoritmus\ 6.53$

$\tilde{g}_{k+1} \coloneqq B_{k+1}^T g_k$

$\eta_{k+1} \coloneqq -B_{k+1}\tilde{g}_{k+1}$

$x_{k+1} \coloneqq algoritmus\ 6.25$

$k \coloneqq k + 1$

if $DelkaKroku < \varepsilon_3$ ***or*** $\|x_{k+1} - x_k\| < \varepsilon_4$ ***or*** $|f(\bar{x}) - f(x_{k+1})| < \varepsilon_5$

break

end if

end while

iv. Výstup:

$\tilde{x} \coloneqq x_k$

$iterace \coloneqq k$

Implementaci uvádíme v Příloze C7. Algoritmus je podobný SDG metodě, v iteračním předpisu metody jsme místo sugbradientu použili rozdíl dvou po sobě jdoucích subgradientů. K výpočtu $\tilde{g}_{k+1}$ dodejme, že matice $R_\kappa(\xi_{k+1})$ je podle věty (6.9) symetrická, a tudíž platí:

$$\tilde{g}_{k+1} = R_\kappa(\xi_{k+1})g_k^* = R_\kappa(\xi_{k+1})B_k^T g_k = \left(B_k R_\kappa^T(\xi_{k+1})\right)^T g_k = \left(B_k R_\kappa(\xi_{k+1})\right)^T g_k = B_{k+1}^T g_k.$$

Zbývá dovysvětlit konkrétní nastavení jednotlivých parametrů. Rozsáhlým testováním N.Z. Shora, N.G. Zhurbenka a dalších vědců byly zjištěny následující optimální hodnoty, které vyhovují širokému spektru funkcí (*Tabulka 18*). Výsledky jejich práce jsme již využili dříve. Uvedené hodnoty nejsou dogma, zvláště v případech, kdy neznáme ani přibližnou polohu minima, lze volit $h > 0{,}1$.

Délka kroku				**Dilatace prostoru**			
h	γ	μ	L	α	p	δ	ν
0,1	0,1	1,25	5	2 ÷ 3	10	1	10

Tabulka 18 – Výchozí optimální hodnoty parametrů.

Je ovšem dobré si uvědomit, že každý zde uvedený algoritmus obvykle vznikl na základě vědecké práce mnoha matematiků, nikoli jediné osoby. Byť s hlavní myšlenkou r-algoritmu přišel právě N.Z. Shor, uplatnění dilatace prostoru bylo známé již dříve, stejně tak variabilní výpočet délky kroku. Hlavní přínos jeho práce spočívá zejména v uvedení dřívějších teoretických poznatků do praxe, jejich vylepšení a optimalizaci výpočetních algoritmů. Nezapomínejme, že tehdy počítače i obor numerická matematika

teprve začínaly. Nejvýkonnější vědecké počítače tehdy uměly řešit úlohy o dimenzích nejvýše 100-200. Dnes můžeme z pohodlí domova řešit úlohy o dimenzi 100-200 milionů.

Příklad 7.2: Optimalizujte funkci z příkladu (6.32) pro různé počáteční aproximace a přesnost řešení. Parametry použijte z *Tabulky18*, koeficient dilatace prostoru volte 2, 2,5 a 3. Průběh výpočtu graficky znázorněte pro jednu počáteční aproximaci. Výsledky analyzujte. Vše bez znalosti hodnoty minima.

$$f(x_1, x_2) = (3x_1^2 + 2x_2^2 - x_1x_2) + 30(|x_1 + x_2| + |x_1 - x_2|).$$

Řešení: Počáteční aproximace a přesnost řešení plynou z následujících tabulek. Implementace řešení je uvedena v Příloze D4. Poloměr okolí pro výpočet subgradientu $\boldsymbol{\zeta = 10^{-4}}$.

ε^1	$(10^{-3}; 10^{-9}; -1; 10^{-4}; -1)$
ε^2	$(10^{-6}; 10^{-18}; -1; 10^{-7}; -1)$
ε^3	$(10^{-9}; 10^{-27}; -1; 10^{-10}; -1)$
ε^4	$(10^{-12}; 10^{-36}; -1; 10^{-14}; -1)$
ε^5	$(10^{-15}; 10^{-45}; -1; 10^{-17}; -1)$

Tabulka 19 – Nastavení přesnosti.

Výpočetní náročnost – počet iterací a **čas uvedený v milisekundách (ms)**.

$\boldsymbol{\alpha = 2,0}$	ε^1	ε^2	ε^3	ε^4	ε^5
$x_0^1 = \begin{pmatrix}5\\5\end{pmatrix}$	0,64	1,3	1,4	1,6	1,7
	28	78	87	96	107
$x_0^2 = \begin{pmatrix}-10\\10\end{pmatrix}$	1,7	1,8	1,9	2,1	2,1
	109	117	127	137	140
$x_0^3 = \begin{pmatrix}10\\0\end{pmatrix}$	0,91	1,2	1,3	1,4	1,5
	60	78	88	97	100
$x_0^4 = \begin{pmatrix}5\\-50\end{pmatrix}$	1,2	1,8	2,0	2,1	2,2
	70	109	119	127	140
$x_0^5 = \begin{pmatrix}20\\-17\end{pmatrix}$	1,1	1,4	1,5	1,7	1,7
	70	89	99	110	120

Tabulka 20 – Výpočetní náročnost pro $\alpha = 2$, dole počet iterací, nahoře čas v milisekundách. Všechny výpočty byly ukončeny pomocí kritéria č. 4 – vzdálenosti dvou po sobě jdoucích aproximací $\|x_{k+1} - x_k\| < \varepsilon_4^i$.

$\boldsymbol{\alpha = 2,5}$	ε^1	ε^2	ε^3	ε^4	ε^5
$x_0^1 = \begin{pmatrix}5\\5\end{pmatrix}$	1,4	2,7	3,1	3,4	3,6
	80	160	189	209	229
$x_0^2 = \begin{pmatrix}-10\\10\end{pmatrix}$	7,7	8,9	9,4	9,7	10,4
	470	560	590	630	658
$x_0^3 = \begin{pmatrix}10\\0\end{pmatrix}$	15,1	15,4	15,6	16,0	16,4
	920	940	969	989	1000
$x_0^4 = \begin{pmatrix}5\\-50\end{pmatrix}$	4,6	4,9	5,1	5,5	5,5
	280	310	320	339	358
$x_0^5 = \begin{pmatrix}20\\-17\end{pmatrix}$	7,0	7,3	7,5	7,9	8,3
	409	430	440	470	509

Tabulka 21 – Výpočetní náročnost pro $\alpha = 2{,}5$, dole počet iterací, nahoře čas v milisekundách. Ve všech případech jsme získali správný výsledek, zelené podbarvení značí dobrou rychlost konvergence (nízký počet iterací), žluté nízkou rychlost konvergence (velký počet iterací). Všechny výpočty byly ukončeny pomocí kritéria č. 4 – vzdálenosti dvou po sobě jdoucích aproximací $\|x_{k+1} - x_k\| < \varepsilon_4^i$.

$\boldsymbol{\alpha = 3,0}$	ε^1	ε^2	ε^3	ε^4	ε^5
$x_0^1 = \begin{pmatrix}5\\5\end{pmatrix}$	13,5	$x_{err}^1 = 10^{12} \cdot [-1{,}7096; 0{,}3918]^T$			
	768	3292			
$x_0^2 = \begin{pmatrix}-10\\10\end{pmatrix}$	$x_{err}^2 = 10^{12} \cdot [0{,}1880; -1{,}1434]^T$				
	1766				
$x_0^3 = \begin{pmatrix}10\\0\end{pmatrix}$	$x_{err}^3 = 10^{13} \cdot [9{,}7057; -2{,}4875]^T$				
	1442				
$x_0^4 = \begin{pmatrix}5\\-50\end{pmatrix}$	$x_{err}^4 = 10^{13} \cdot [2{,}8935; -0{,}1170]^T$				
	3102				
$x_0^5 = \begin{pmatrix}20\\-17\end{pmatrix}$	40,1	$x_{err}^5 = 10^{13} \cdot [-1{,}0050; -0{,}0961]^T$			
	2450	2762			

Tabulka 22 – Výpočetní náročnost pro $\alpha = 3$, dole počet iterací, nahoře čas v milisekundách. Žluté podbarvení značí správný výsledek, ale nízkou rychlost konvergence. Červené podbarvení značí chybný výsledek – výpočet se sice vždy zastavil, ale konvergoval k nesprávnému řešení (uvádíme chybné řešení a počet iterací).

Z výsledků testování je patrné, že koeficient prostorové dilatace α nelze zvyšovat bez rozmyslu. Obecně platí, že vysoký koeficient α, velká počáteční délka kroku a současně požadavek na velmi malou chybu mohou vést k chybnému výsledku. Pro úplnost zopakujme – metoda ve všech testech zastavila, avšak v červeně podbarvených polích jsme nezískali správný výsledek (vzdálenost od správného řešení je řádově 10^{12} až 10^{13}).

Důsledek 7.3:

Parametry pro nastavení výpočtu nejsou nezávislé veličiny. Vždy je potřeba přihlédnout ke konkrétní výchozí situaci. Pokud máme o cenové funkci, respektive o poloze minima málo informací, vyplatí se volit nižší hodnoty koeficientu prostorové dilatace.

Zastavme se ještě u *Tabulky 22.* Nejdříve dodejme, že oba výpočty se správným výsledkem a všechny výpočty pro x_0^2 byly ukončeny pomocí kritéria číslo 4 – vzdálenosti dvou po sobě jdoucích aproximací $\|x_{k+1} - x_k\| < \varepsilon_4^i$. Ostatní výpočty byly ukončeny pomocí kritéria číslo 1 – velikosti subgradientu $\|g_k\| < \varepsilon_1^i$. Tabulka v sobě ukrývá hned několik zajímavostí.

Nejdříve si všimněme, že pro chybný výsledek je typický vyšší počet iterací (navíc zde nejde o miliony iterací, ale pouze o nižší jednotky tisíců). Na tomto příkladu je dobře vidět následující skutečnost.

Důsledek 7.4:

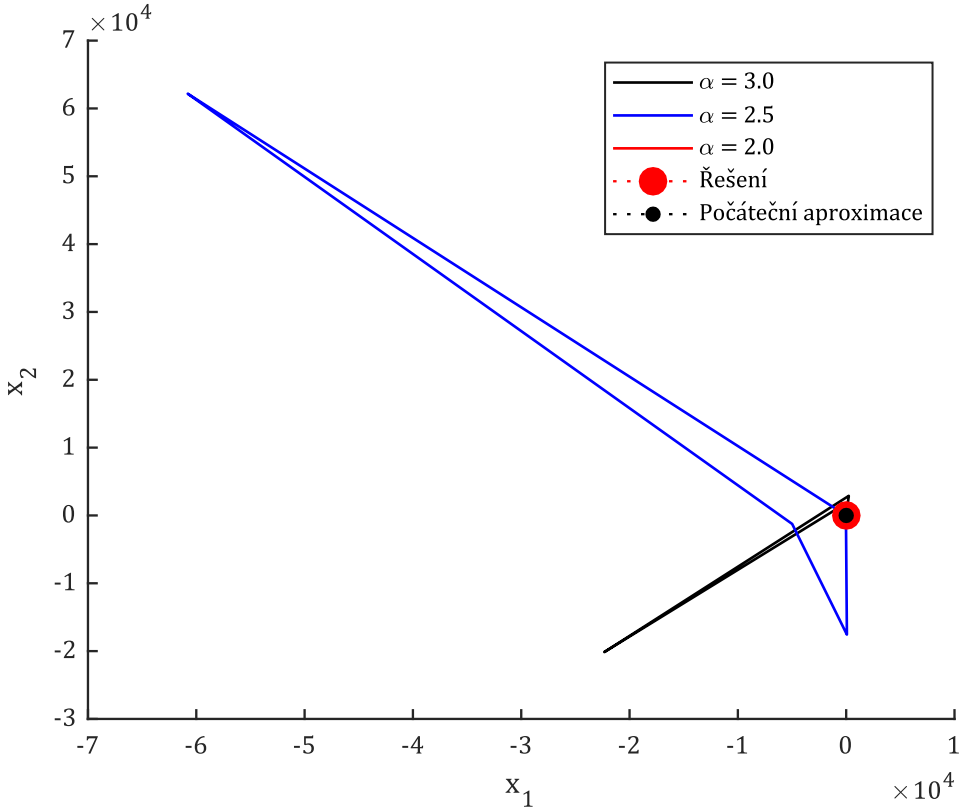
Vyšší počet iterací při provádění výpočtu neznamená pouze delší výpočetní čas, ale i mnohem větší prostor pro šíření numerických chyb. Právě z těchto důvodů se vždy snažíme používat algoritmy (nebo jejich nastavení) takové, že počet iterací je minimální. Navíc relativně vysoký počet iterací může ukazovat na nesprávnost řešení.

Dále se zaměříme na **chybné výpočty**. Pro každou danou počáteční aproximaci x_0^i výpočet konvergoval k výsledku x_{err}^i, a to při stejném počtu iterací bez ohledu na požadovanou přesnost řešení, což je velice zvláštní. Pro konkrétní počáteční aproximaci výpočet sice probíhá vždy stejným způsobem včetně jeho zastavení po dosažení jedné z ukončovacích podmínek, avšak tyto podmínky jsou pro různé přesnosti nastaveny různě. Všechny chybné výpočty pro danou počáteční aproximaci navíc skončili po dosažení stejného typu ukončovací podmínky, jejíž hodnota se ale řádově liší. Čekali bychom zvyšující se počet iterací se zvyšující se přesností, ale přesto výpočet vykonal vždy stejný počet iterací. **Proč?**

Nejkritičtějším místem výpočtu je jednoznačně matice B. Upravme hodnoty parametrů $p = 10 \rightarrow 5$ (počet iterací do optimalizace matice B), $h = 0{,}1 \rightarrow 0{,}5$ (výchozí délka kroku) a pokus pro $\alpha = 3$ opakujme. Výsledky uvádíme v *Tabulce 23*. Úpravou vstupních parametrů r-algoritmu jsme docílili mnohem lepších výsledků pro vyšší hodnotu koeficientu prostorové dilatace $\alpha = 3{,}0$. Nicméně i přesto dosažené výsledky byly horší než pro $\alpha = 2{,}0$ (porovnejte s *Tabulkou 20*). Stejným postupem bychom mohli optimalizovat výpočet i pro jiné hodnoty koeficientu α. To si ovšem žádá systematičtější přístup k problému, proto jsme numerické experimenty umístili do vlastní kapitoly.

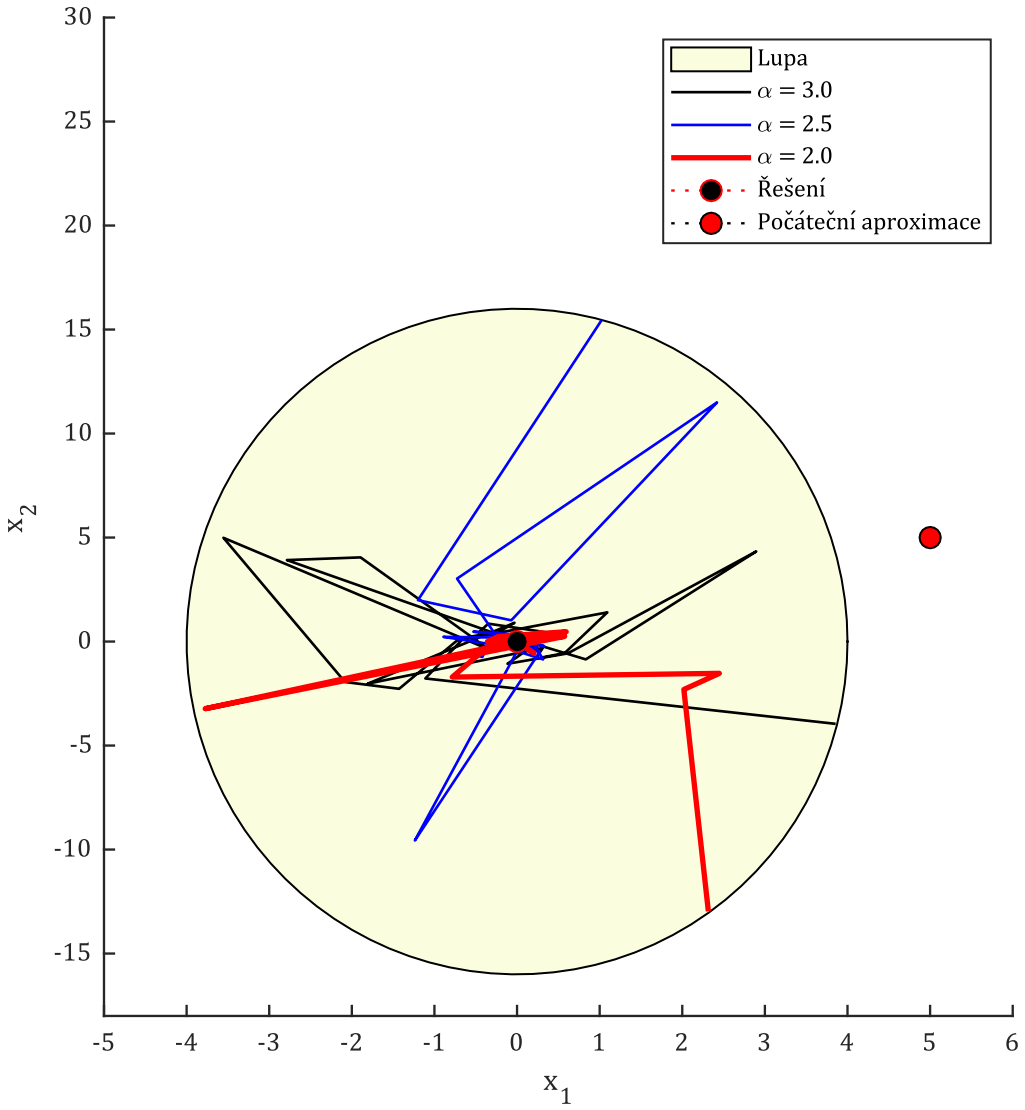
$\boldsymbol{\alpha = 3,0}$ $\boldsymbol{p = 5}$ $\boldsymbol{h = 0,5}$	ε^1	ε^2	ε^3	ε^4	ε^5
$x_0^1 = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$	2,2	2,2	2,3	2,6	2,9
	109	120	128	153	168
$x_0^2 = \begin{pmatrix} -10 \\ 10 \end{pmatrix}$	6,1	13,4	13,6	13,7	13,8
	380	763	768	778	789
$x_0^3 = \begin{pmatrix} 10 \\ 0 \end{pmatrix}$	1,3	6,5	6,7	6,7	7,2
	73	330	335	350	363
$x_0^4 = \begin{pmatrix} 5 \\ -50 \end{pmatrix}$	2,3	2,4	2,6	2,7	3,1
	134	148	156	168	191
$x_0^5 = \begin{pmatrix} 20 \\ -17 \end{pmatrix}$	4,1	12,5	12,8	13,2	13,4
	200	669	698	708	725

Tabulka 23 – Výpočetní náročnost pro $\alpha = 3$ s úpravou parametrů p a h.



Obrázek 19 – Srovnání průběhu výpočtu pro počáteční aproximaci $x_0 = (5,5)^T$ a různé koeficienty dilatace prostoru.

Nyní dokončeme úlohu, chybí nám grafické znázornění průběhu výpočtu. Počáteční aproximaci zvolíme $x_0^1 = (5;5)^T$, přesnost ε^1 a trasujeme pro koeficienty $\alpha = 2{,}0, 2{,}5$ a $3{,}0$ s $p = 5$ a $h = 5$. Průběh výpočtu je znázorněn na *Obrázku 19*. Podívejme se pořádně na rozsah hodnot na jednotlivých osách, bohužel z tohoto důvodu splývá počáteční aproximace s řešením. Výpočet probíhá na relativně velké oblasti vůči počáteční aproximaci $[5;5]^T$ a řešení $[0;0]^T$. Na obrázku není vidět červený graf výpočtu pro $\alpha = 2{,}0$, protože tento probíhá na mnohem menší oblasti. Lepší pohled na průběh výpočtu v okolí řešení vykresluje *Obrázek 20* – přiblížení předchozího obrázku v okolí počátku.



Obrázek 20 – Detailnější pohled na průběhu výpočtu pro počáteční aproximaci $x_0 = (5{,}5)^T$ a různé koeficienty dilatace prostoru..

△

7.4 Konvergence r-algoritmu, modifikace r-mikro a r-alfa

V této kapitole nastíníme důkaz konvergence r-algoritmu.

Věta 7.5: ***O konvergenci r-algoritmu***

Nechť funkce $f: \mathbb{R}^n \to \mathbb{R}$ je lokálně lipschitzovsky spojitá na množině $\mathbb{R}^n$. Nechť f je navíc zdola omezená. Pak algoritmus (7.1) pro libovolnou počáteční aproximaci konverguje k bodu $x_{min}^L \in \mathbb{R}^n$, ve kterém funkce f nabývá lokálního minima.

Pozor, věta (7.5) pouze říká, že r-algoritmus konverguje k *nějakému* lokálnímu minimu. Pokud existuje právě jedno lokální minimum na $\mathbb{R}^n$ a je současně i globálním minimem, pak r-algoritmus konverguje k tomuto minimu (korektní úloha). Pokud existuje více minim, pak algoritmus najde jedno z nich, ale dopředu nevíme, které to bude (a vůbec se nemusí jednat o minimum s nejmenší funkční hodnotou). Zde opět vidíme, proč musíme před samotným výpočtem provést separaci lokálních minim.

Celý důkaz věty (7.5) je velmi složitý a zdlouhavý, proto uvádíme pouze jeho nástin. Zájemcům o celý důkaz můžeme doporučit literaturu [9]. Důkaz probíhá ve dvou krocích. Nejdříve *zidealizujeme* původní r-algoritmus tak, aby pro každou aproximaci řešení platilo

$$f(x_{k+1}) \leq f(x_k). \tag{7.6}$$

Podmínku (7.6) splňuje **algoritmus r-mikro**, označujeme jej $r_\mu(\alpha)$. Koeficient dilatace prostoru $\alpha > 1$ již známe, a koeficient μ volíme tak, aby platilo $0 \leq \mu < 1$. V algoritmu (7.1) pak nevolíme libovolný subgradient $g_k \in G_k$, ale takový, který splňuje

$$(g_k^*, \tilde{g}_k) \leq \mu \|g_k^*\| \|\tilde{g}_k\|. \tag{7.7}$$

Tento r-mikro algoritmus nám zaručí, že funkční hodnoty v jednotlivých aproximacích $x_0, x_1, \ldots, x_k$ budou tvořit nerostoucí posloupnost $f(x_0) \geq f(x_1) \geq \ldots \geq f(x_k)$. Pro $\mu = 0$ získáme zvláštní případ r-mikro algoritmu – **r-alfa algoritmus**, značíme jej $r_0(\alpha)$.

Nalezení subgradientu g_k splňujícího nerovnost (7.7) není vůbec triviální. Této podmínce je navíc nutné přizpůsobit konstrukci celého algoritmu (7.1). Výhodou je ovšem nerostoucí posloupnost $f(x_0) \geq f(x_1) \geq \ldots \geq f(x_k)$, pro kterou je snadnější dokázat konvergenci. Následně ukážeme, že uvedené zidealizování (*občasný* růst funkčních hodnot v aproximacích řešení) nevede ke ztrátě konvergence.

7.5 Výpočetní náročnost r-algoritmu

U každého algoritmu potřebujeme určit alespoň jeho časovou a paměťovou náročnost (obvykle stačí jejich odhad). V matematice dále používáme termíny rychlost konvergence a řád konvergence. Rychlost konvergence výpočetní metody je nepřímo úměrná její časové náročnosti. Řád konvergence r-algoritmu neexistuje, není splněna podmínka, aby se aproximace řešení v každé iteraci přiblížila přesnému řešení. Více o této tematice najdeme v literatuře [8].

Nechť $n \in \mathbb{N}$ je dimenze minimalizační úlohy. **Paměťová náročnost M** implementace algoritmu (7.1) je dána především pamětí nutnou pro uložení tří matic (každá n^2 jednotek) a šesti vektorů (n jednotek). Můžeme ji vyjádřit vztahem

$$M(n, D) = (3n^2 + 6n + c) \cdot D, \tag{7.8}$$

kde $c \in \mathbb{N}$ je konstanta (počet jednorozměrných proměnných) a $D \in \mathbb{N}$ je paměťová náročnost použitého datového typu (například v bytech). Pro Matlab je výchozí typ *double* (dvojitá přesnost), který zabírá pro celá i desetinná čísla stejně 8 bytů (tedy 64 bitů). Podstatný je obvykle odhad náročnosti

$$\mathcal{O}_M(n) = n^2, \tag{7.9}$$

tudíž se jedná o polynomiální paměťovou náročnost. V případě stolního počítače pro dimenzi $n \leq 100$ je použitá paměť zanedbatelná vzhledem k velikosti současných pamětí RAM.

Vyjádřit **časovou náročnost T** je podstatně složitější, můžeme ji však zapsat ve tvaru

$$T(\alpha, \varepsilon, n) = P(\alpha, \varepsilon) \cdot t(n, h, \gamma, \mu, L), \tag{7.10}$$

kde P je počet iterací nutných k dosažení přesnosti ε a t je výpočetní čas jedné iterace. Je zřejmé, že počet iterací (pro konkrétní funkci) záleží především na požadované maximální chybě a koeficientu dilatace prostoru, a že výpočet jedné iterace závisí velmi složitě na mnoha parametrech. Počet iterací ovšem dopředu stanovit neumíme, proto celou časovou náročnost T zjistíme experimentálně pro několik cenových funkcí.

8 Numerické experimenty

V této kapitole si ukážeme (kromě samotné minimalizace), jak stanovit časovou náročnost výpočtu, maximální přesnost výpočtu a řadu dalších užitečných ukazatelů. Zájemci o hlubší pochopení tematiky mohou pokračovat studiem elektronické přílohy této kapitoly, která volně navazuje na podkapitolu 8.3.

Pokud není uvedeno jinak, vždy uvažujeme přesnost

$$\varepsilon = (10^{-6}; 10^{-18}; -1; 10^{-7}; -1).$$

Upozorněme, že uvedená přesnost slouží primárně k zastavení výpočtu, tedy že výpočet v daném smyslu dosáhl uvedené přesnosti. Skutečnou odchylku aproximace řešení od přesného řešení stanovujeme vždy až zpětně (v ukázkových úlohách polohu přesného řešení obvykle známe).

Domluvme se, že časovou náročnost vyjádříme v počtu iterací. Pak můžeme vztah (7.10) zjednodušit

$$T(\alpha, \varepsilon) = P(\alpha, \varepsilon), \tag{8.1}$$

kde $P(\alpha, \varepsilon)$ je počet iterací r-algoritmu v závislosti na přesnosti a koeficientu dilatace prostoru. Ten je však pevně daný, a tedy **časovou náročnost** určíme pouze jako funkci přesnosti ε (v iteracích)

$$T(\varepsilon) = P(\varepsilon). \tag{8.2}$$

Použití vztahu (8.2) je ovšem komplikované, proto raději využijeme standardní sady testovacích úloh se znalostí bodu minima $\bar{x}$ a funkční hodnoty v něm $f(\bar{x})$. Pro absolutní chyby výpočtu platí

$$\Delta_X = \|\bar{x} - \tilde{x}\|, \tag{8.3}$$

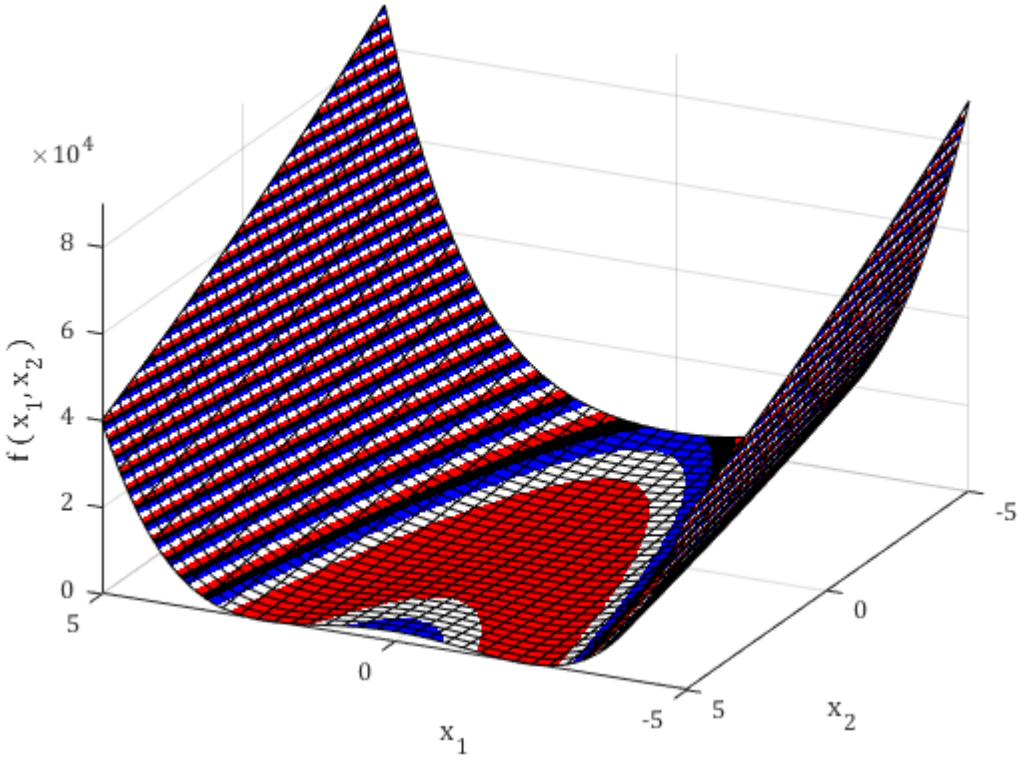
$$\Delta_F = |f(\bar{x}) - f(\tilde{x})|, \tag{8.4}$$

kde $\tilde{x}$ představuje aproximaci řešení v dané iteraci. Vyjádření chyby ve tvaru (8.3) použijeme v úlohách, kde hledáme primárně přesné souřadnice bodu minima – *lokační úlohy*. Oproti tomu chybu typu (8.4) použijeme například při minimalizaci nákladů výroby, kdy je požadováno primárně skutečné minimum cenové funkce (nákladů na výrobu). My v každé iteraci i určíme obě tyto chyby. **Rychlost konvergence a maximální přesnost výpočtu** vyčteme z lineární interpolace grafu funkce

$$\xi: \mathbb{N} \to \mathbb{R}, \xi(i) = \Delta_i, i = 1,2, \ldots \tag{8.5}$$

Pozor, nehledáme přímo minimum funkce ξ, jde nám o konvergenci. V rámci vhodných úloh otestujeme závislost chyby (8.3) na volbě počáteční aproximace x_0 a koeficientu dilatace prostoru α. Nakonec dodejme, že v kapitole 8.1 budeme velmi podrobně analyzovat dosažené výsledky. V případě ostatních funkcí budeme již postupovat rychleji. Nyní můžeme přistoupit k samotným experimentům.

8.1 Rosenbrock



Obrázek 21 – Graf Rosenbrockovy funkce s barevně odlišenými vrstevnicemi funkčních hodnot.

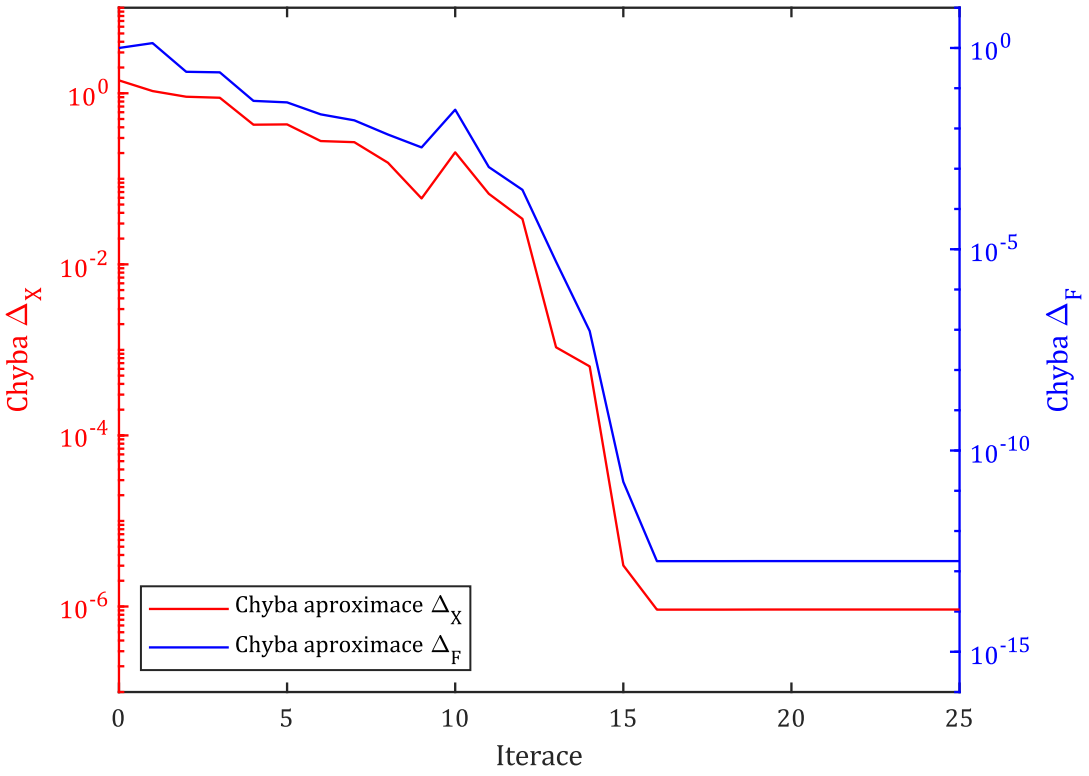
Funkční předpis	$f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$						
Dimenze úlohy	2						
Přesné řešení	$\bar{x} = (1; 1)$			$f(\bar{x}) = 0$			
Počáteční aprox.	$x_0 = (0; 0)$						
Délka kroku				**Dilatace prostoru**			
h	γ	μ	L	α	p	δ	ν
0,1	0,1	1,5	10^5	2,5	10	1	10

Tabulka 24 – Zadání Rosenbrockovy funkce.

Rosenbrockova funkce je sice spojitá a má spojité všechny parciální derivace libovolného řádu, ale její charakteristiky jsou mnohem bližší nehladkým funkcím. Jedná se o ryze testovací funkci – používá se pro zjištění parametrů optimalizační metody jako rychlost konvergence, přesnost, spolehlivost a dalších.

8.1.1 Výpočetní náročnost a maximální přesnost výpočtu

Nejdříve určíme **výpočetní náročnost a maximální přesnost**. Volat algoritmus pro každou přesnost ε není příliš rozumné. Budeme postupovat opačně – algoritmus necháme běžet a v každé iteraci odečteme přesnost výpočtu. Pro naše potřeby jsme naprogramovali testovací verzi r-algoritmu, kterou najdete v Příloze C8. V každé iteraci zaznamenáme aproximaci řešení $\tilde{x}$ a $f(\tilde{x})$ a chyby řešení dané vztahy (8.3) a (8.4). Průběh minimalizace z hlediska aproximačních chyb zobrazuje graf na *Obrázku 22*.



Obrázek 22 – Graf závislosti chyby aproximace na počtu iterací.

Typ chyby	Minimální hodnota	V kolikáté iteraci	Výsledná	Celkem iterací
Δ_X	$9{,}1886 \cdot 10^{-7}$	17	$9{,}2157 \cdot 10^{-7}$	25
Δ_F	$1{,}7821 \cdot 10^{-13}$	17	$1{,}7902 \cdot 10^{-13}$	

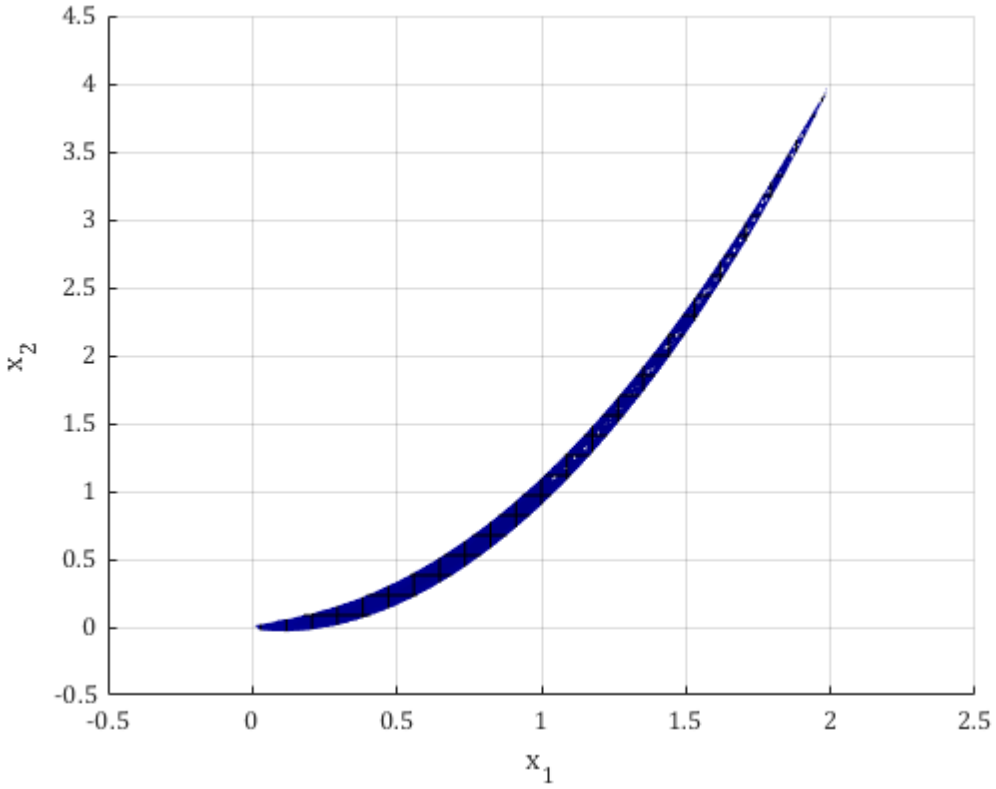
Tabulka 25 – Maximální přesnost minimalizace Rosenbrockovy funkce.

Počet provedených iterací je velice nízký. **Všimněme si desáté iterace** – oba typy chyby zde vzrostly. Zajímavější je zvýšení Δ_F, protože znamená **vzrůst funkční hodnoty**. V desáté iteraci k minimalizaci ve skutečnosti nedošlo. Tento nedostatek r-algoritmu odstraňuje jeho varianta r-mikro (kapitola 7.4).

Z grafu je rovněž patrné, že minimalizace probíhala přímočaře. Největší pokles chyb nastal mezi 10-15 iterací, minimální hodnotu obou chyb jsme naměřili v 17té iteraci, avšak až do konce výpočtu došlo pouze k minimálnímu nárůstu chyby (relativně v řádu jednotek promile). Algoritmus se sám zastavil z důvodu nulové délky kroku po 25ti iteracích. Naměřené výsledky shrneme do *Tabulky 26*. Dodejme, že rozdílná rychlost poklesu chyb Δ_X a Δ_F je daná primárně charakterem funkce, nikoli metody.

8.1.2 Testování stability výpočtu – 1. část

První část úkolu spočívá v **testování stability výpočtu pro různé počáteční aproximace**. Nejdříve se podívejme na graf funkce omezené shora hodnotou $f_{max} = 1$, *Obrázek 23*.



Obrázek 23 – Část grafu Rosenbrockovy funkce – pohled shora.

Vidíme, že hledaný bod minima jistě patří do oblasti

$$\bar{x} \in M = (-0{,}5; 2{,}5) \times (-0{,}5; 4{,}5).$$

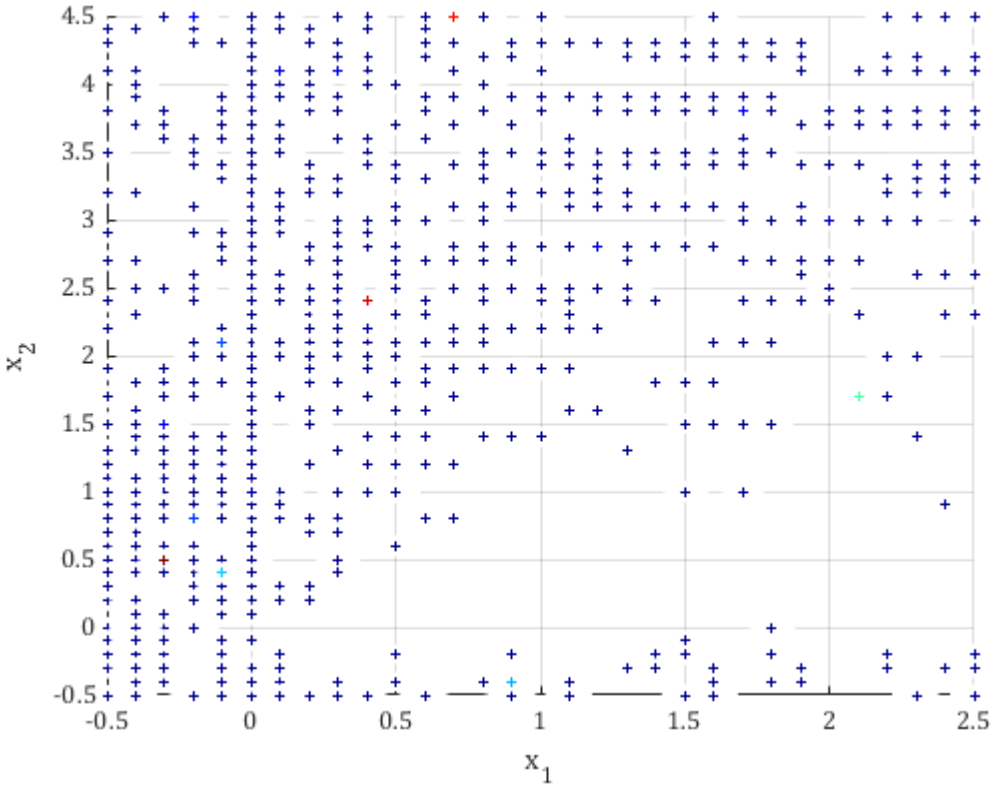
Nad oblastí M použijeme metodu sítě, jejíž uzly budou představovat počáteční aproximace. Uvažujme síť S jako množinu uzlů

$$S = \{(1 + 0{,}1j, 1 + 0{,}1k) \in \mathbb{R}^2 \mid j \in [-15; 15], k \in [-15; 35]\}.$$

Použijeme metodu `Shor.m` s parametry z *Tabulky 24,* kterou voláme pomocí obyčejného a paralelního cyklu `for/parfor` pro všechny počáteční aproximace. Metoda vždy vrací počet iterací a aproximaci minima, ze které vypočteme obě chyby. Hodnoty zaznamenáme do předem připravené struktury.

Nastal problém! Ačkoli jsme využili výrazného omezení oblasti, kde testujeme, i paralelismu, výpočet se zdá být nekonečný... Odhalme příčinu komplikací. Po řadě pokusů jsme přišli na následující zjištění.

Problém nastal při výpočtu optimální délky kroku v algoritmu (6.25). Výpočet probíhá v každé iteraci a za určitých okolností může běžet velice dlouho. Délku výpočtu lze ovlivnit parametrem *L*. Čím nižší hodnotu parametru *L* zvolíme, tím častěji proběhne změna výchozí délky kroku a tím se celý výpočet urychlí. Současně ovšem jeho relativně nízké hodnoty vedou k vyšší nestabilitě výpočtu – chybnému výsledku, a stejně zcela nezabrání velmi dlouhým výpočtům. **Pozor, hodnota parametru *L* ovlivňuje délku jedné iterace Shorova r-algoritmu, nikoli jejich počet**.



Obrázek 24 – Počáteční aproximace, ve kterých výpočet selhal.

Řešením je doplnit algoritmus (6.25) o kontrolu – určit časový limit pro hledání optimální délky kroku. Za tuto činnost je odpovědný cyklus `while`, který modifikujeme (limit 5 sekund). Tento postup nijak neovlivní dříve dokázanou konvergenci. Časovou limit 15 sekund přidáme i do hlavního algoritmu (7.1). Nyní máme zajištěno, že výpočet skončí v rozumné době. Uvedený způsob monitorování se v praxi velice často využívá. Například společnost Intel používá mikročip s názvem *Watch Dog* umístěný přímo na základní desce počítače pro kontrolu správného běhu kritických aplikací.

Nakonec spustíme náš test znovu. *Obrázek 24* ukazuje počáteční aproximace, ve kterých výpočet selhal. Tento stav samozřejmě není vůbec ideální, nejen kvůli počtu selhání. Zejména nevíme, zdali jsme získali smysluplný výsledek. Řešením problému je alespoň přibližná znalost oblasti, ve které se nachází bod minima. Tu ovšem (v tomto případě) ale známe. Průměr oblasti M je

$$R_M = \|(-0{,}5; -0{,}5) - (2{,}5; 4{,}5)\| = \|(-3;\ -5)\| = \sqrt{34},$$

poloměr

$$r_M = \frac{1}{2} R_M = \frac{1}{2}\sqrt{34},$$

a střed má souřadnice

$$S_M = (1; 2).$$

To znamená, že pro přesné řešení $\bar{x}$ musí platit podmínka

$$\|\bar{x} - \tilde{x}\|^2 \leq \frac{17}{2}. \tag{8.6}$$

Odfiltrujme z výsledků experimentu aproximace $\tilde{x}$, pro které není splněna nerovnost (8.6), a zjistíme, že pro chybu Δ_X^{max} všech zbylých aproximací platí

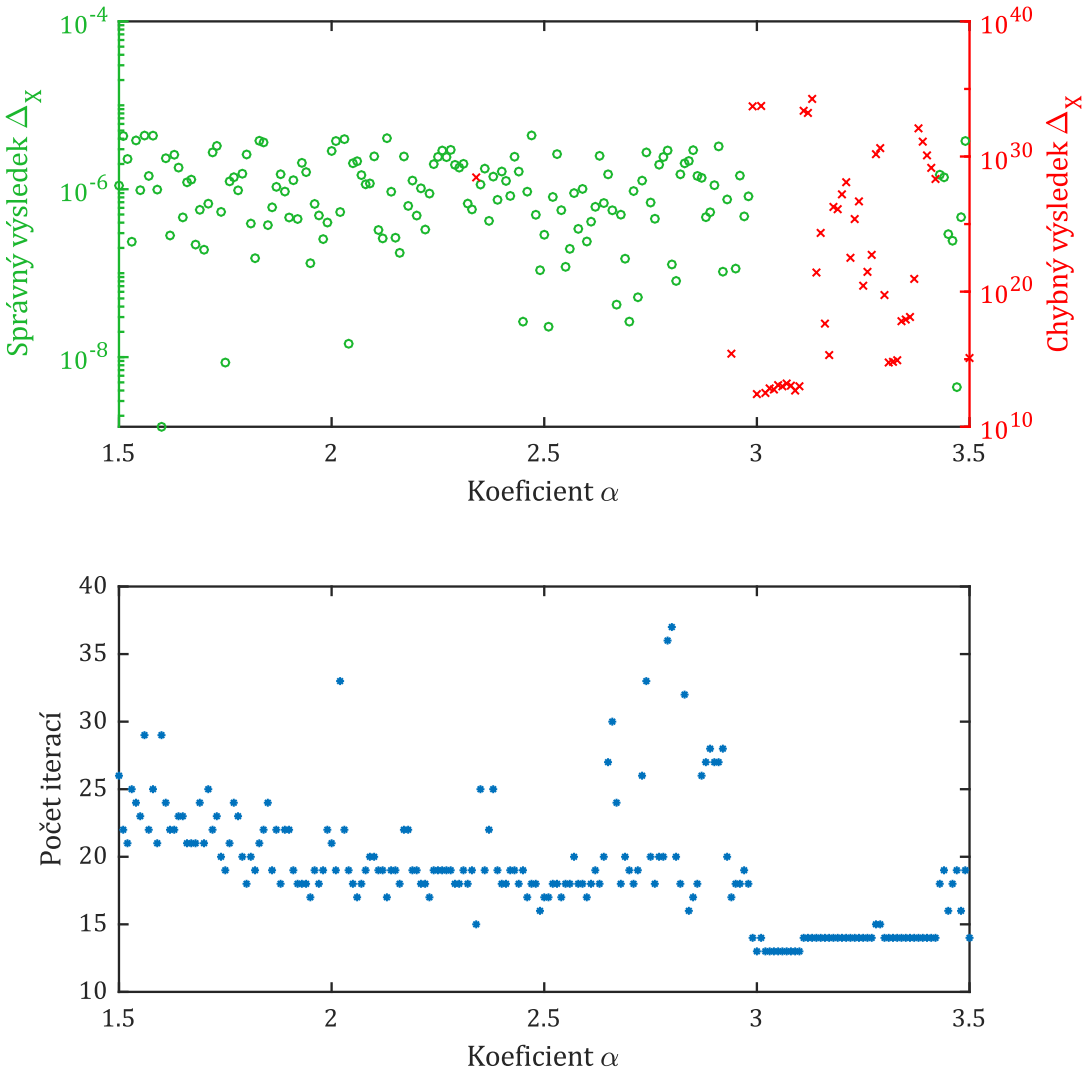
$$\Delta_X^{max} = 6{,}8389 \cdot 10^{-6}.$$

Jinak řečeno, se znalostí alespoň přibližné polohy bodu minima jsme schopni odfiltrovat veškeré chybné výpočty. Právě proto **je důležité znát alespoň přibližnou polohu bodu minima**. Pokud identifikujeme chybný výpočet, můžeme optimalizaci spustit znovu s jinou počáteční aproximací. Samozřejmě právě popsaná úvaha nemusí nutně vést ke správnému výsledku, avšak v praxi se ukazuje, že chybný výpočet v naprosté většině (běžných) situací konverguje k zjevně nesmyslnému výsledku. Stejný potenciál pro identifikaci chybného výpočtu má rovněž **znalost funkční hodnoty v bodu minima**.

Nabízí se otázka, zdali můžeme chybný výpočet rozpoznat i jiným způsobem, například podle počtu provedených iterací. Počet iterací pro jednotlivé počáteční aproximace se pohybuje v rozmezí od 5 do 179 a nezjistili jsme korelaci mezi počtem iterací a správností výpočtu. Výsledky našeho experimentu tedy ukazují, že nikoli.

Druhá část úkolu spočívá v **testování stability výpočtu pro různé koeficienty dilatace prostoru α**. Koeficient α volíme v rozmezí od 1,5 do 3,5 s krokem 0,01. Výsledky měření ukazuje *Obrázek 25 nahoře*. Algoritmus pracuje správně do koeficientu $\alpha_{max} = 2{,}93$ (s jedinou výjimkou). Poté již vidíme sérii chybných výsledků, přičemž opět platí, že tyto lze snadno rozpoznat a odstranit, pokud známe alespoň přibližnou polohu bodu minima. Chybný výpočet naopak nelze spolehlivě identifikovat

podle počtu iterací, byť v tomto případě již můžeme sledovat slabou korelaci obou veličin (při chybných výpočtech je počet iterací nižší), jak ukazuje *Obrázek 25 dole*. Výsledek experimentu částečně potvrzuje dříve uvedené závěry – pro vyšší hodnoty koeficientu α je výpočet méně stabilní, avšak kratší výpočetní dobu jsme nezaznamenali. Kód řešení pro první a druhou podkapitolu je uveden v Příloze D5.



Obrázek 25 – Nahoře: Závislost chyby aproximace na koeficientu prostorové dilatace. Chyba aproximace výpočtu se správným výsledkem (označeno zeleně) se pohybuje řádově od 10^{-8} do 10^{-5}. Chyba aproximace chybného výpočtu (označeno červeně) je diametrálně odlišná, řádově větší než 10^{10}. Dole: Závislost počtu iterací na koeficientu dilatace prostoru.

8.1.3 Modifikace r-algoritmu – autodetekce a oprava chybných výpočtů

Až doteď jsme se zabývali minimalizací primárně z akademického hlediska. Reálná situace se odlišuje tím, že je v ní menší prostor pro ideály. Ideálem nazvěme požadavek na rychlé, přesné, jednoduché a nenáročné řešení současně. V praxi obvykle musíme volit mezi řešením rychlým a přesným. Řešením máme na mysli algoritmus (program) řešící daný problém.

Jak jsme již zmínili, r-algoritmus je vnímán jako rychlý a nenáročný algoritmus pro minimalizaci funkcí spíše nižších dimenzí. V případě vyšších dimenzí $n > 10^4$ je překážkou v jeho použití vysoká paměťová náročnost běžného zápisu matice B. Vysoká rychlost r-algoritmu ovšem současně znamená možný problém s menší přesností, což se nám potvrdilo v minulé části textu. Řešením je přídavný algoritmus pro detekci (pravděpodobně) chybných výsledků, ideálně implementovaný přímo do r-algoritmu.

Tím se otevírá téma reimplementace algoritmů (6.25) a (7.1). Provedeme nezbytné úpravy pro dosažení výše uvedených cílů.

Algoritmus 6.25: Na **vstupu** metody `AproxKoerc2` (oproti `AproxKoerc`) ve struktuře `Data` přibyl sedmý parametr `tMax` pro maximální čas [s] běhu cyklu `while`. **Nemá vliv na důkaz konvergence.**

Algoritmus 7.1: Na **vstupu** metody `Shor2` (oproti `Shor`) přibyl čtvrtý parametr `Rizeni` ve formátu

```
Rizeni = {tMax6_25; tMax7_1; Opakovani; DX; DF; StIter; Stred}.
```

Parametr	**Význam**	
`tMax6_25`	Maximální doba běhu cyklu `while` v algoritmu (6.25).	`tMax > 0` nebo `tMax = inf`
`tMax7_1`	Maximální doba běhu algoritmu (7.1).	
`Opakovani`	Maximální počet pokusů o minimalizaci s různými počátečními aproximacemi.	
`DX`, `DF`	Maximální velikost chyby Δ_X (8.3), respektive chyby Δ_F (8.4).	
`StIter`	Od které iteraci se má provádět kontrola výše uvedených parametrů.	
`Stred`	Souřadnice bodu, vůči kterému měříme chybu Δ_X.	
Poznámka: Všechny parametry jsou povinné s výjimkou parametru `Stred`, pokud současně není zadána chyba `DX`. V případě zadání `DF` metodě předáme hodnotu v bodu minima pomocí struktury `Funkce`. Pokud parametr `DX` nebo `DF` nechceme použít, nastavíme jeho hodnotu na -1. V případě hodnoty parametru `Opakovani > 1` musí alespoň jedna z chyb splňovat `DX` > 0 nebo `DF` > 0.		

Tabulka 26 – Přehled změn pro vstup modifikovaného r-algoritmu.

Kontrolu na velikost chyb `DX` a `DF` provedeme vždy na konci cyklu `for` i `while`. V případě překročení limitů se algoritmus sám restartuje a použije novou počáteční aproximaci (zvolí pseudonáhodně).

Algoritmus 7.1: Na **výstupu** metody `Shor2` zjednodušíme hlášení, proč algoritmus skončil (`Kod`).

Kod	**Význam**
100	Řádný běh a dokončení metody **bez restartu**.
101	Řádný běh a dokončení metody **s restartem**.
200	Běh metody byl ukončen **časovým limitem**, ale výsledek **splňuje kritéria** pro velikost chyby Δ_X a Δ_F. Má vliv na důkaz konvergence.
400	**Chybný výsledek** – metoda provedla maximální počet opakování minimalizace, avšak výsledek stále **nesplňuje kritéria** pro velikost chyby Δ_X nebo Δ_F. Výsledku přiřadíme hodnotu $\tilde{x} = (\infty, \infty, \dots, \infty)$.

Tabulka 27 – Přehled změn pro výstup modifikovaného r-algoritmu.

Reimplementaci metod uvádíme v Příloze C9, Příloze C10 a Příloze C11. V *Tabulce 28* pak použité hodnoty nových parametrů. Parametr `DX` volíme $DX = 10r_M$, Parametr `DF` podle minimalizované funkce, jako parametr `Stred` volíme střed oblasti, ve které se nachází bod minima. Pro obecné úlohy lze volit počátek a vyšší hodnotu `DX`. Kód řešení pro třetí a čtvrtou podkapitolu je uveden v Příloze D6.

tMax6_25	tMax7_1	Opakovani	DX	DF	StIter	Stred
0,100 s	2,000 s	10	$5\sqrt{34}$	-1	10	(1; 2)

Tabulka 28 – Hodnoty nově použitých parametrů.

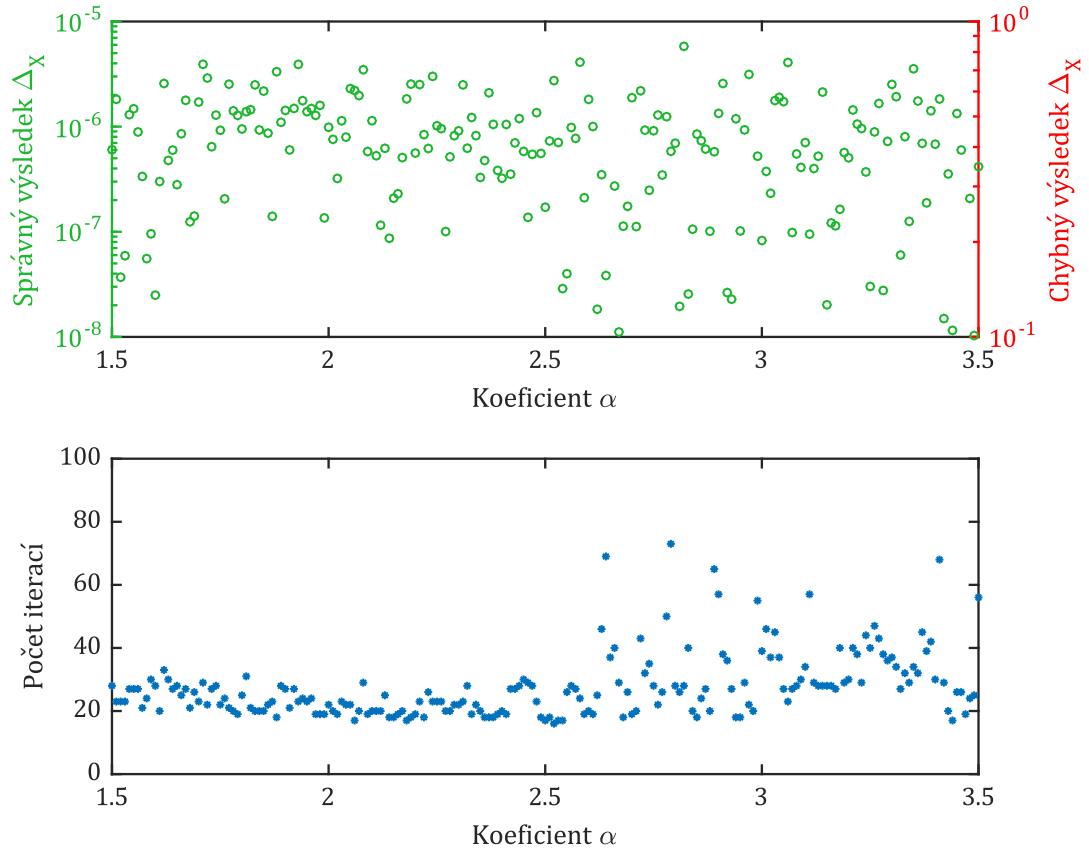
8.1.4 Testování stability výpočtu – 2. část

Pomocí nové implementace r-algoritmu provedeme stejné výpočty jako v kapitole 8.1.2. Nejdříve se zaměříme na srovnání dosažených výsledků – *Tabulka 29*. Je zřejmé, že upravená metoda `Shor2` je nesrovnatelně spolehlivější, pouze pro 0-1 počátečních aproximací jsme nezískali správný výsledek, přičemž všechny chybné výsledky metoda správně označila jako neplatné. Oproti tomu původní metoda má chybných 37 % výsledků, a navíc je neumí označit. Ačkoli modifikovaná metoda provádí opakované výpočty s nutností vyhodnotit více podmínek, výpočetní čas je téměř trojnásobně kratší.

Obrázek 26 je obdobou *Obrázku 25*, avšak pro žádnou hodnotu koeficientu prostorové dilatace nedošlo k chybě. Pro koeficient $\alpha > 2{,}5$ je počet iterací vyšší a značně kolísá, výpočet je méně stabilní.

Měřená veličina	Metoda	
	Původní	**Modifikovaná**
Celkem minimalizací	1581	1581
Neoznačené chybné výsledky	590	0
Označené chybné výsledky	0	0
Celkový výpočetní čas	31,8 s	11,6 s
Průměrný čas na 1 minimalizaci	20,1 ms	7,3 ms
Celkový počet iterací	30234	42913
Průměrný počet iterací na 1 minimalizaci	19,1	27,1
Průměrná chyba správného výsledku	$1{,}1921 \cdot 10^{-6}$	$1{,}2060 \cdot 10^{-6}$

Tabulka 29 – Srovnání výsledků původní a modifikované metody. Hodnoty pro modifikovanou metodu jsou průměrem z deseti měření.



Obrázek 26 – Modifikovaná metoda. Nahoře: Závislost chyby aproximace na koeficientu prostorové dilatace. Chyba aproximace výpočtu se správným výsledkem (označeno zeleně) se pohybuje řádově od 10^{-8} do 10^{-5} . Chybný výpočet (červeně) nebyl zaznamenán. Dole: Závislost počtu iterací na koeficientu dilatace prostoru.

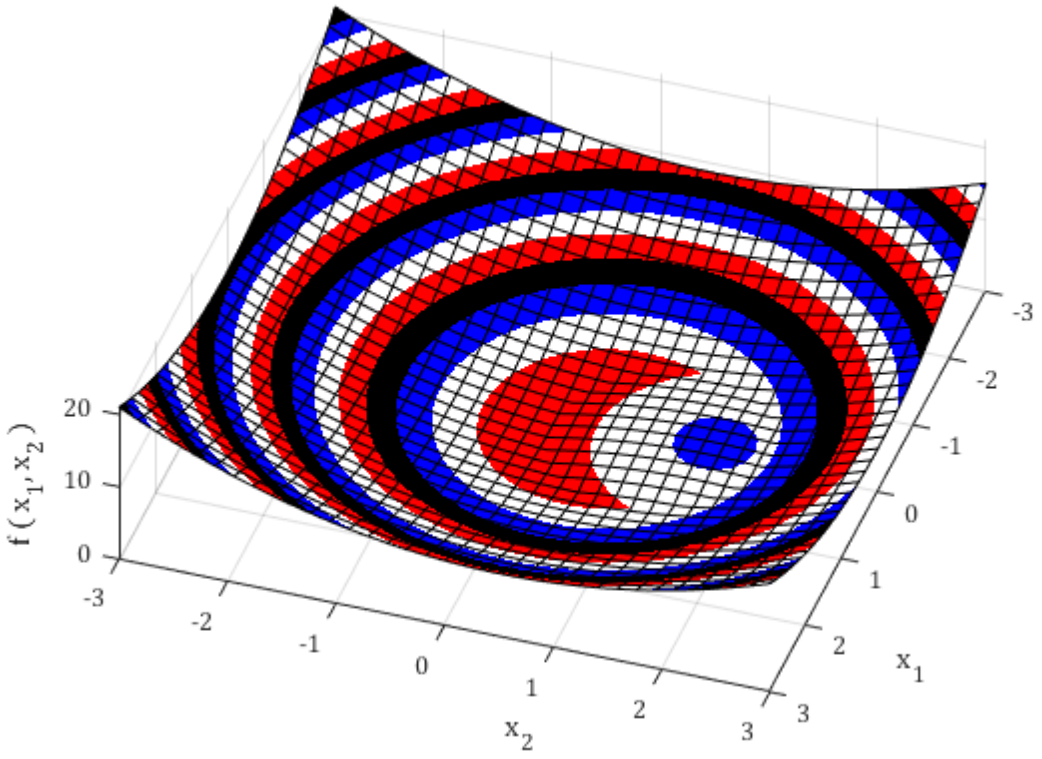
8.1.5 Shrnutí experimentu

Rosenbrockova funkce patří k obtížně optimalizovatelným, přesto se nám povedlo upravit r-algoritmus tak, aby byl úspěšný prakticky ve 100 % minimalizací pro libovolnou počáteční aproximaci a koeficient prostorové dilatace α. Navíc upravený algoritmus byl třikrát rychlejší.

Modifikovaná metoda je vhodná pro použití v aplikacích, které vyžadují spolehlivost. Předpokládá však znalost alespoň odhadu oblasti, ve které se nachází bod minima. Nespornou výhodou modifikované metody je možnost použití širšího rozsahu hodnot minimalizačních parametrů, aniž by došlo k selhání. **Nasazení metody je tedy vhodné i v případech, kdy o průběhu minimalizované funkce nemáme dostatek informací** (za parametr `Stred` dosadíme počátek soustavy souřadnic a volíme vyšší `DX`).

Na závěr ještě dodejme, že naše modifikace je poměrně univerzální. Pomocí parametrů můžeme metodu vhodně nastavit pro široké spektrum kritérií, jako například rychlost, spolehlivost, přesnost i další.

8.2 Půlměsíc

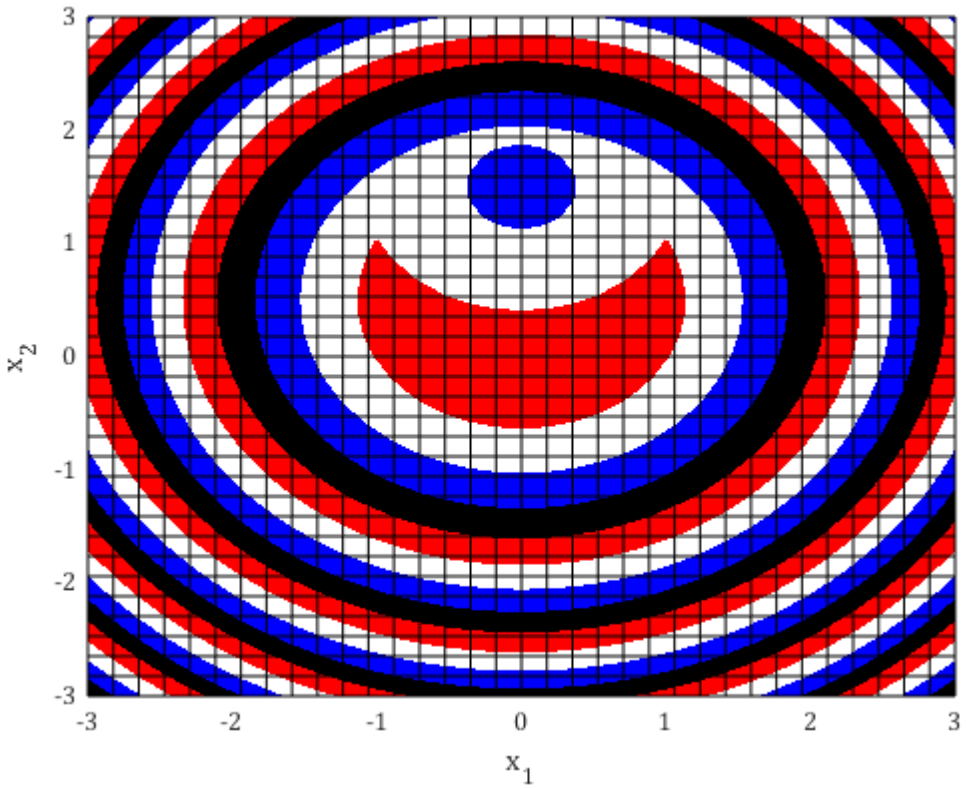


Obrázek 27 – Graf funkce Půlměsíc s barevně odlišenými vrstevnicemi funkčních hodnot.

Funkční předpis	$f(\boldsymbol{x}) = \max\{x_1^2 + (x_2 - 1)^2 + x_2 - 1, -x_1^2 - (x_2 - 1)^2 + x_2 + 1\}$						
Dimenze úlohy	2						
Přesné řešení	$\bar{x} = (0; 0)$			$f(\bar{x}) = 0$			
Počáteční aproximace	$x_0 = (-2; -2)$						
Délka kroku				**Dilatace prostoru**			
h	γ	μ	L	α	p	δ	ν
0,1	0,1	1,5	100	2,5	10	1	10
Řízení náročnosti a běhu výpočtu							
`tMax6_25`	`tMax7_1`	`Opakovani`	`DX`	`DF`	`StIter`	`Stred`	
0,100 s	2,000 s	10	$30\sqrt{2}$	-1	10	(0; 0)	

Tabulka 30 – Zadání minimalizace funkce Půlměsíc.

Tato analytická funkce získala název podle tvaru svých vrstevnic připomínajícího půlměsíc (pro kóty blížící se nule). Pro odhad oblasti, ve které leží bod minima, je vhodnější pohled shora – *Obrázek 28*. Odhad je uveden v *Tabulce 31*. Dále postupujeme stejně jako u Rosenbrockovy funkce. Kód řešení pro první i druhou podkapitolu je uveden v Příloze D7.

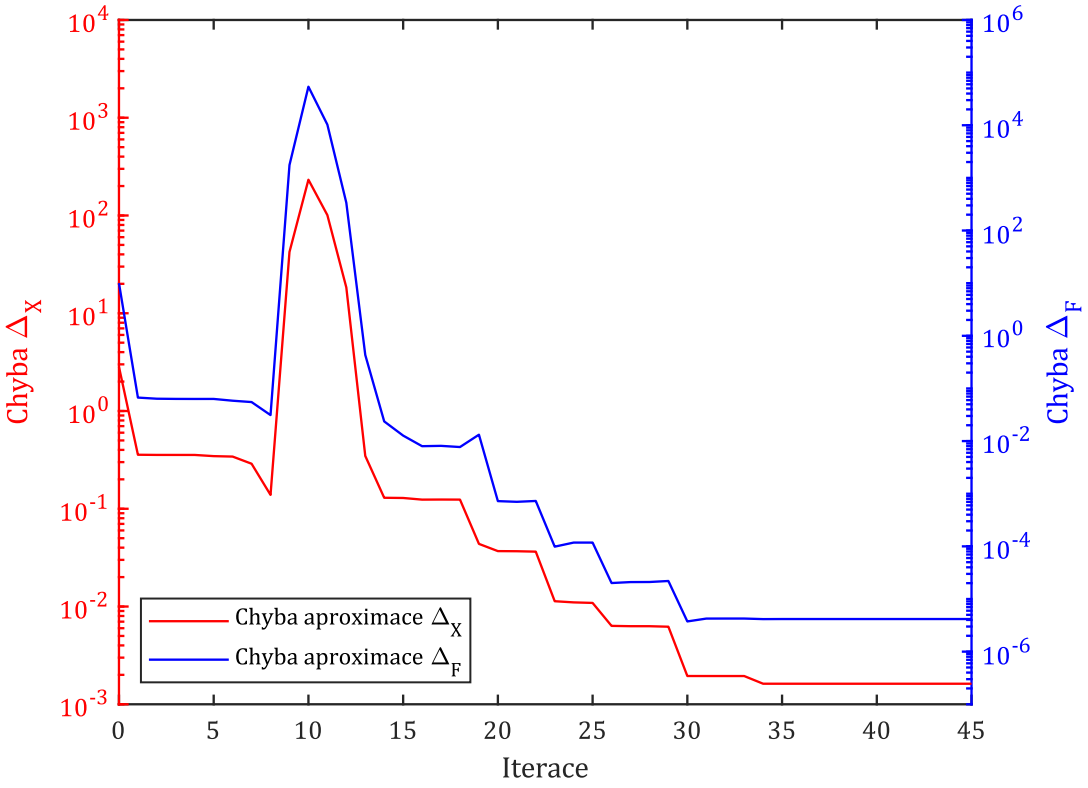


Obrázek 28 – Graf funkce Půlměsíc – pohled shora

Parametr	Oblast M	Poloměr r_M	Střed S_M
Hodnota	$(-3;\ 3) \times (-3;\ 3)$	$3\sqrt{2}$	$(0;\ 0)$

Tabulka 31 – Odhad oblasti, ve které leží bod minima.

8.2.1 Výpočetní náročnost a maximální přesnost výpočtu



Obrázek 29 – Graf závislosti chyby aproximace na počtu iterací.

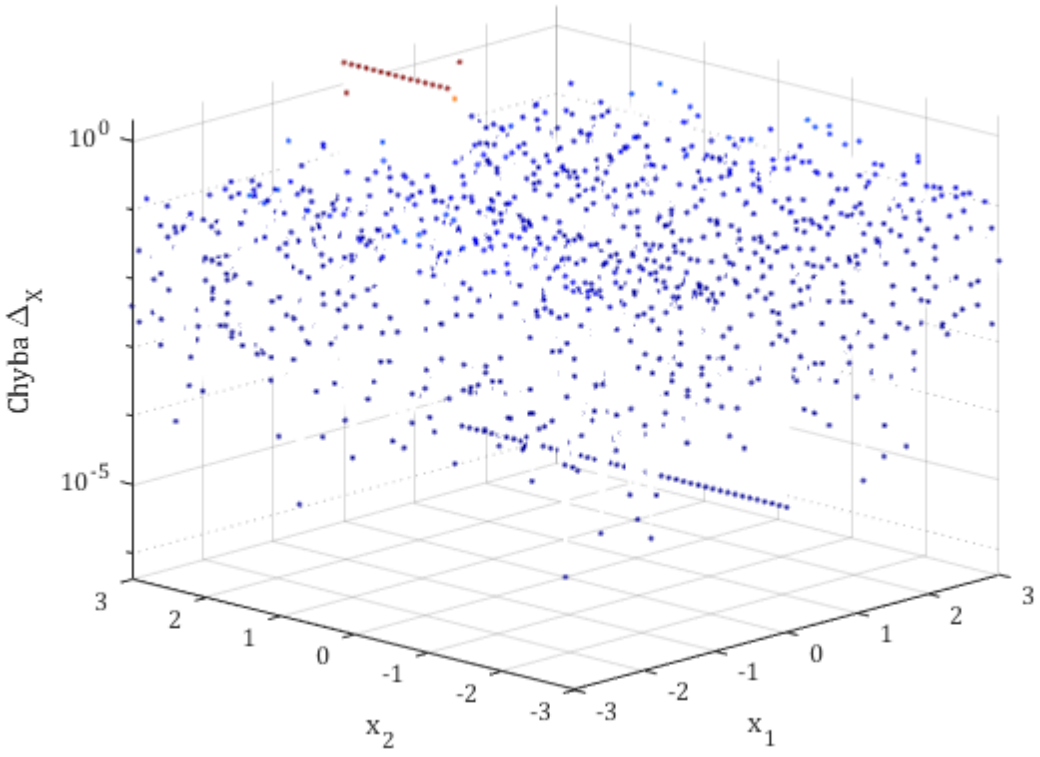
Graf na *Obrázku 29* ukazuje, že v desáté iteraci prudce vzrostly obě chyby. Výpočet ukončila nulová délka transformovaného subgradientu. *Tabulka 32* ukazuje dosaženou maximální přesnost výpočtu. Z grafu je také patrné, že oba typy chyb po dosažení svého minima se zvýšili pouze nepatrně.

Typ chyby	**Minimální hodnota**	**V kolikáté iteraci**	**Výsledná**	**Celkem iterací**
Δ_X	$1{,}6225 \cdot 10^{-3}$	35	$1{,}6225 \cdot 10^{-3}$	45
Δ_F	$3{,}7484 \cdot 10^{-6}$	30	$3{,}7484 \cdot 10^{-6}$	

Tabulka 32 – Maximální přesnost minimalizace funkce Půlměsíc.

8.2.2 Zvýšení spolehlivosti výpočtu – parametr `opakovani`

Provedeme test stability výpočtu pro tuto funkci. **Pro 224 počátečních aproximací z celkových 3721 jsme nezískali správné řešení**. Modifikovaná metoda `Shor2` sice všechna tato chybná řešení označila, ale 6,02 % nepoužitelných výsledků je příliš. Závislost chyby Δ_X na počáteční aproximaci zobrazuje *Obrázek 30*. Všimněme si, že maximální chyba nastala pro $x_1 = 0$ a $x_2 \geq 1{,}5$.



Obrázek 30 – Graf závislosti chyby výpočtu na počáteční aproximaci.

Náš problém má ale jednoduché řešení – zvýšíme počet opakování hledání minima v případě selhání (volíme různé počáteční aproximace v okolí původní). Je ovšem otázkou, jak se jeho zvýšení projeví na celkovém výpočetním čase, a zdali skutečně dojde k snížení počtu chyb výpočtu. Parametr `opakovani` nastavíme podle *Tabulky 33* a změříme výpočetní čas a počet chyb (procentuální úspěšnost výpočtu). Pro úplnost statistiku doplníme průměrnou chybou funkční hodnoty $\overline{\Delta}_F$, jejím mediánem$\tilde{\Delta}_F^{0,5}$ a mediánem chyby $\tilde{\Delta}_X^{0,5}$. Výsledky porovnáme s původním nastavením výpočtu.

Měření	1	2	3	4	5	6	7
`Opakovani`	10	20	50	100	200	500	1000

Tabulka 33 – Zadání počtu opakování výpočtu v případě jeho selhání.

Počet opakování	Δ_X^{max}	Označené chyby		Průměr na 1 minimalizaci			
		Počet	%	Čas [ms]	$\overline{\Delta}_F$	$\tilde{\Delta}_F^{0,5}$	$\tilde{\Delta}_X^{0,5}$
10	2,000	224	6,02	4,64	0,0773	$3{,}92 \cdot 10^{-5}$	0,0079
20	2,000	59	1,59	5,04	0,0842	$3{,}51 \cdot 10^{-5}$	0,0069
50	2,000	8	0,22	5,08	0,0766	$3{,}34 \cdot 10^{-5}$	0,0064
100	2,000	1	0,03	5,24	0,0749	$3{,}38 \cdot 10^{-5}$	0,0065
200	2,000	< 1	0,02	5,33	0,0882	$3{,}25 \cdot 10^{-5}$	0,0063
500	2,000	≈ 0	< 0,01	5,25	0,0883	$3{,}22 \cdot 10^{-5}$	0,0059
1000	2,000	≈ 0	≈ 0	5,22	0,0903	$3{,}44 \cdot 10^{-5}$	0,0065

Tabulka 34 – Výsledky experimentu pro různý počet opakování výpočtu v případě selhání. Provedli jsme 3721 minimalizací pro každé měření.

Tabulka 34 ukazuje výsledky měření. **Počet chyb prudce klesá** se zvyšujícím se počtem opakování výpočtu v případě jeho selhání. **Opakování výpočtu je automatické** – vlastnost naší implementace. Přičemž současně průměrná **časová náročnost výpočtu se významně nezvyšuje**, což je pro nás zásadní. Zde však buďme obezřetní – v případě jiných cenových funkcí vůbec nemusí platit, že časová náročnost výrazně neporoste.

Nicméně maximální velikost chyby je stále $\Delta_X^{max} = 2$ pro libovolný počet opakování. Tento problém naše modifikace r-algoritmu odstranit neumí, protože je totiž primárně určena k odstranění divergence směrem k nekonečnu. Kdežto u funkce Půlměsíc máme problém s pomalou konvergencí a konvergencí k jinému bodu, než je skutečné minimum, avšak tento bod se nachází v blízkosti minima.

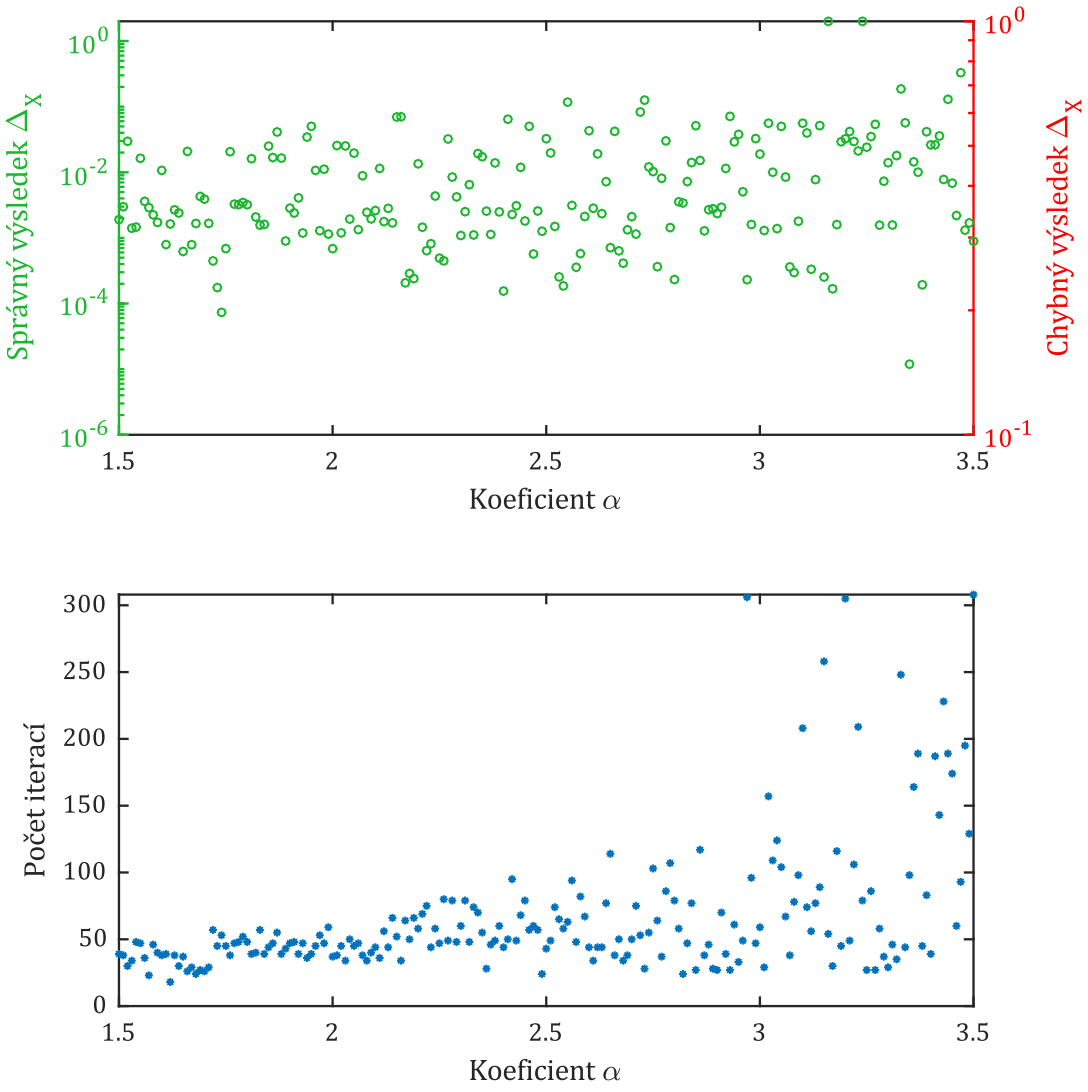
Závěr

Modifikovaná implementace r-algoritmu opět ukázala mnohem vyšší rychlost, spolehlivost a také širší možnosti použití, zejména pak v automatizovaných procesech. Vhodným nastavením metody dokážeme předcházet chybným výpočtům.

Nevýhodou použití r-algoritmu při minimalizaci funkce *Půlměsíc* je nižší průměrná přesnost aproximace řešení, avšak medián chyby je o dva řády nižší.

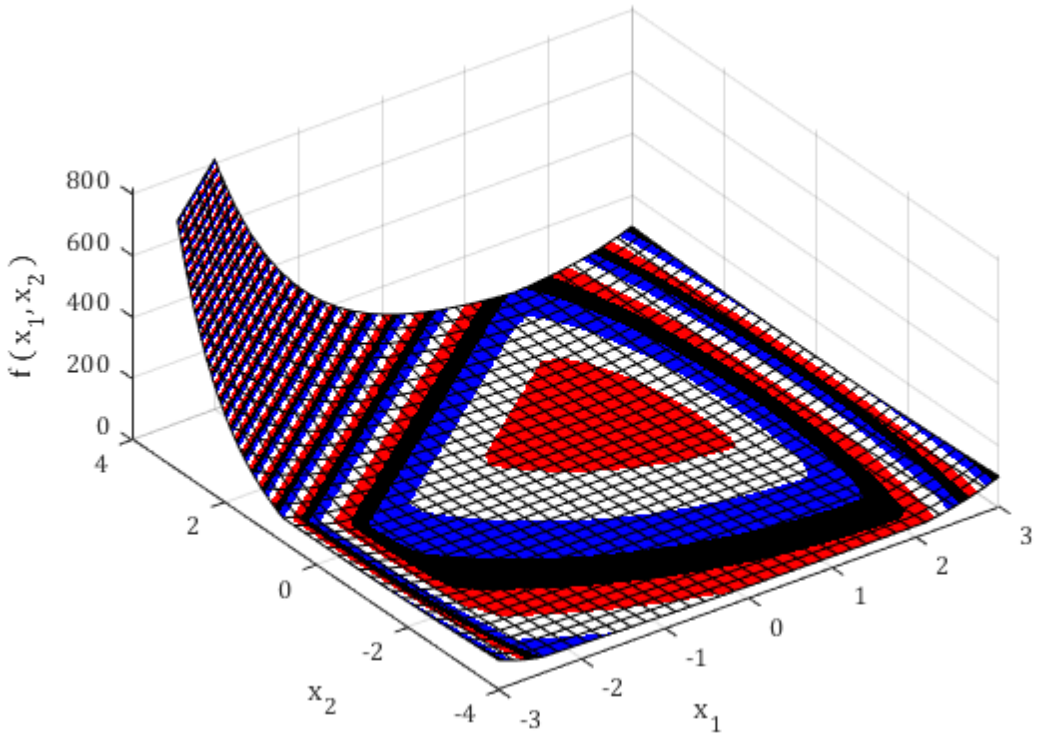
8.2.3 Závislost výpočtu na koeficientu prostorové dilatace α

Nyní otestujeme závislost chyby a počtu iterací na volbě koeficientu prostorové dilatace α. Výsledky zobrazuje graf na *Obrázku 31*. Pro vyšší hodnoty parametru $\alpha > 2{,}5$ je počet iterací vyšší a značně závislý na volbě počáteční aproximace.



Obrázek 31 – Nahoře: Závislost chyby aproximace na koeficientu prostorové dilatace. Chyba aproximace výpočtu se správným výsledkem (označeno zeleně) se pohybuje řádově od 10^{-5} do 10^{0} . Chybný výpočet (červeně) nebyl zaznamenán. Dole: Závislost počtu iterací na koeficientu dilatace prostoru.

8.3 Charalambous-Bandler



Obrázek 32 – Graf funkce Charalambous-Bandler s barevně odlišenými vrstevnicemi funkčních hodnot.

Funkční předpis	$f(\boldsymbol{x}) = \max\{x_1^4 + x_2^2, (2-x_1)^2 + (2-x_2)^2, 2e^{-x_1+x_2}\}$						
Dimenze úlohy	2						
Přesné řešení	$\bar{x} = (1;1)$			$f(\bar{x}) = 2$			
Počáteční aproximace	$x_0 = (0;2)$						
Délka kroku				**Dilatace prostoru**			
h	γ	μ	L	α	p	δ	ν
0,1	0,1	1,5	2500	2,5	10	1	10
Řízení náročnosti a běhu výpočtu							
`tMax6_25`	`tMax7_1`	`Opakovani`	`DX`	`DF`	`StIter`	`Stred`	
0,100 s	2,000 s	10	25	-1	10	(0,5; 1)	

Tabulka 35 – Zadání Charalambousovy-Bandlerovy funkce.

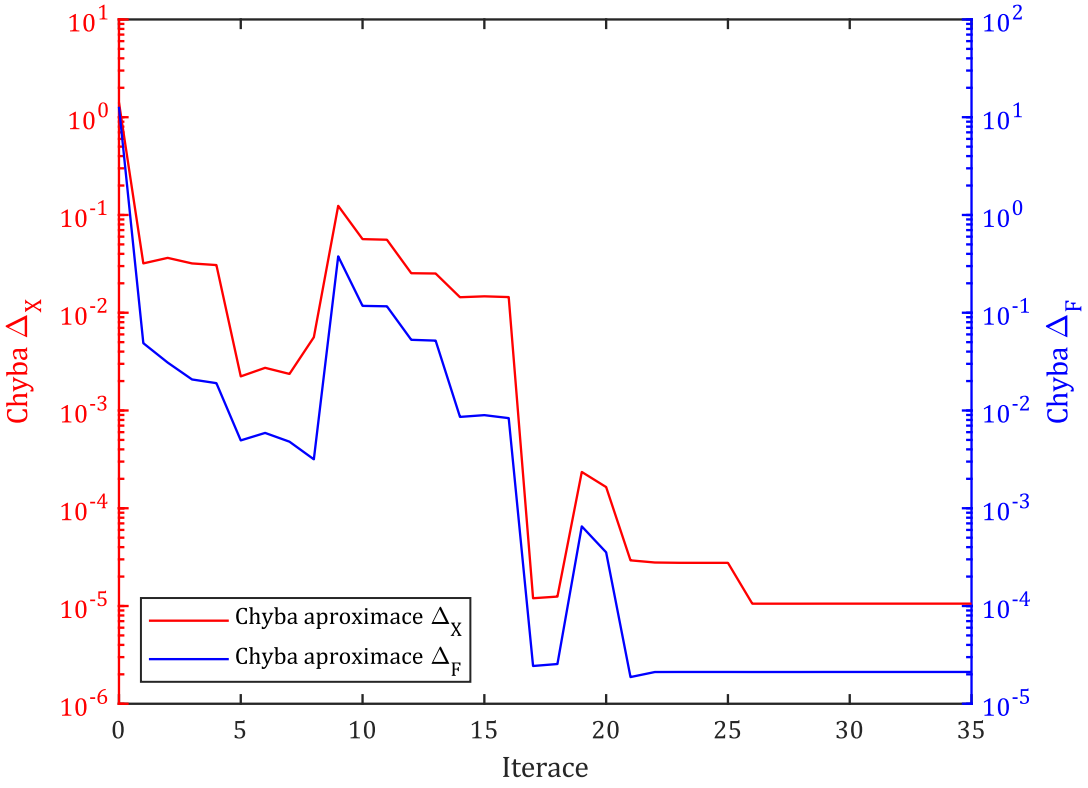
Funkce získala název podle dvou kanadských autorů – Christakise Charalambouse a Johna W. Bandlera. Používá se především v souvislosti s modelováním bezdrátových širokopásmových sítí včetně wi-fi.

Nejdříve z grafu odhadneme oblast, ve které leží mod minima (*Tabulka 36*). Dále postupujeme stejně jako v předchozích úlohách.

Parametr	Oblast M	Poloměr r_M	Střed S_M
Hodnota	$(-1;\ 2) \times (-1;\ 3)$	2,5	$(0{,}5;\ 1)$

Tabulka 36 – Odhad oblasti, ve které leží bod minima.

8.3.1 Výpočetní náročnost a maximální přesnost výpočtu



Obrázek 33 – Graf závislosti chyby aproximace na počtu iterací.

Typ chyby	**Minimální hodnota**	**V kolikáté iteraci**	**Výsledná**	**Celkem iterací**
Δ_X	$1{,}0562 \cdot 10^{-5}$	26	$1{,}0569 \cdot 10^{-5}$	35
Δ_F	$1{,}8750 \cdot 10^{-5}$	21	$2{,}1150 \cdot 10^{-5}$	

Tabulka 37 – Maximální přesnost minimalizace Charalambousovy-Bandlerovy funkce.

Výpočetní náročnost je vidět na *Obrázku 33*, maximální přesnost v *Tabulce 37.* Výpočet byl ukončen z důvodu nulové délky kroku. Kód řešení pro první podkapitolu je uveden v Příloze D8.

8.3.2 Závislost charakteristik výpočtu na parametru L

Uvažujme minimalizaci Charalambousovy-Bandlerovy funkce, oblast M z *Tabulky 37*, síť počátečních aproximací na oblasti M s pevným krokem 0,10 a koeficient α od 1,00 do 4,00 s pevným krokem 0,10. Provedeme měření pro čtyři hodnoty parametru L (*Tabulka 38)*, který určuje, jak často se aktualizuje výchozí délka kroku v iteraci.

Měření	1	2	3	4
L	10	20	50	100

Tabulka 38 – Hodnoty použitých parametrů.

Testujeme závislost nejdůležitějších parametrů výpočtu (spolehlivost, přesnost a rychlost) v závislosti na koeficientu prostorové dilatace α. Výsledek testu nám poskytne lepší náhled na výkonnost metody.

Zajímají nás následující údaje pro každé α:

- **Spolehlivost výpočtu C_α**

$$S = \frac{C^*}{C},$$

  kde C^* je počet výsledků, pro které platí $\Delta_X \leq r_M$, a C je počet všech výsledků (počet minimalizací pro jednu hodnotu koeficientu α).

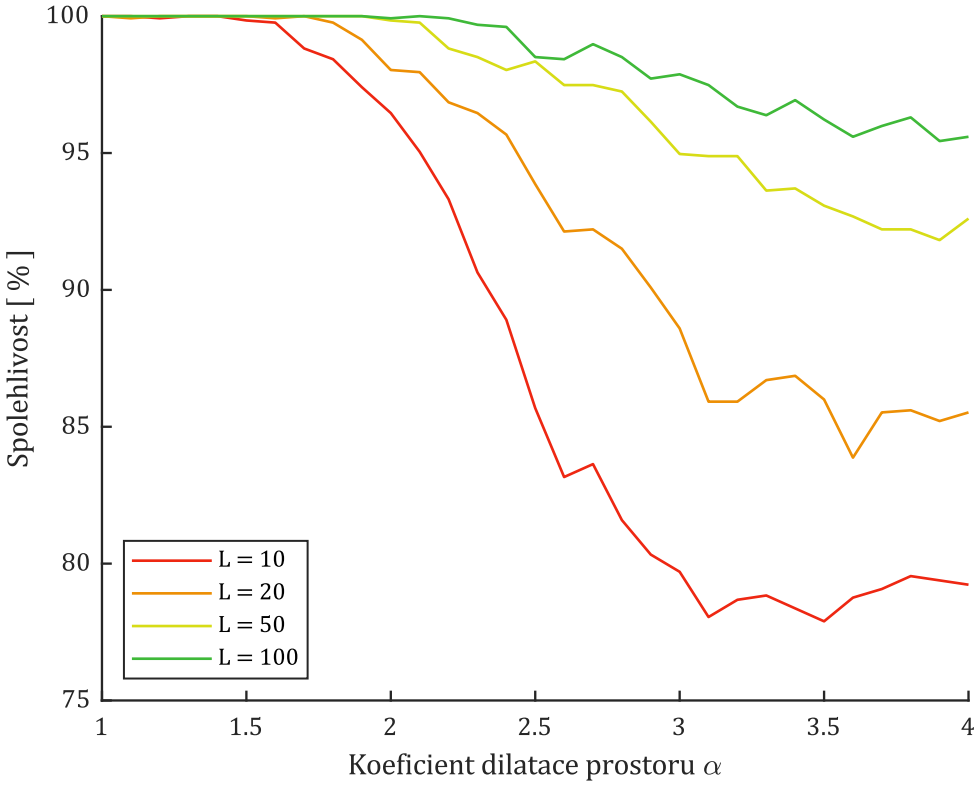
- **Přesnost výpočtu P_α**

$$P_\alpha = \frac{\sum \Delta_X^*}{C^*},$$

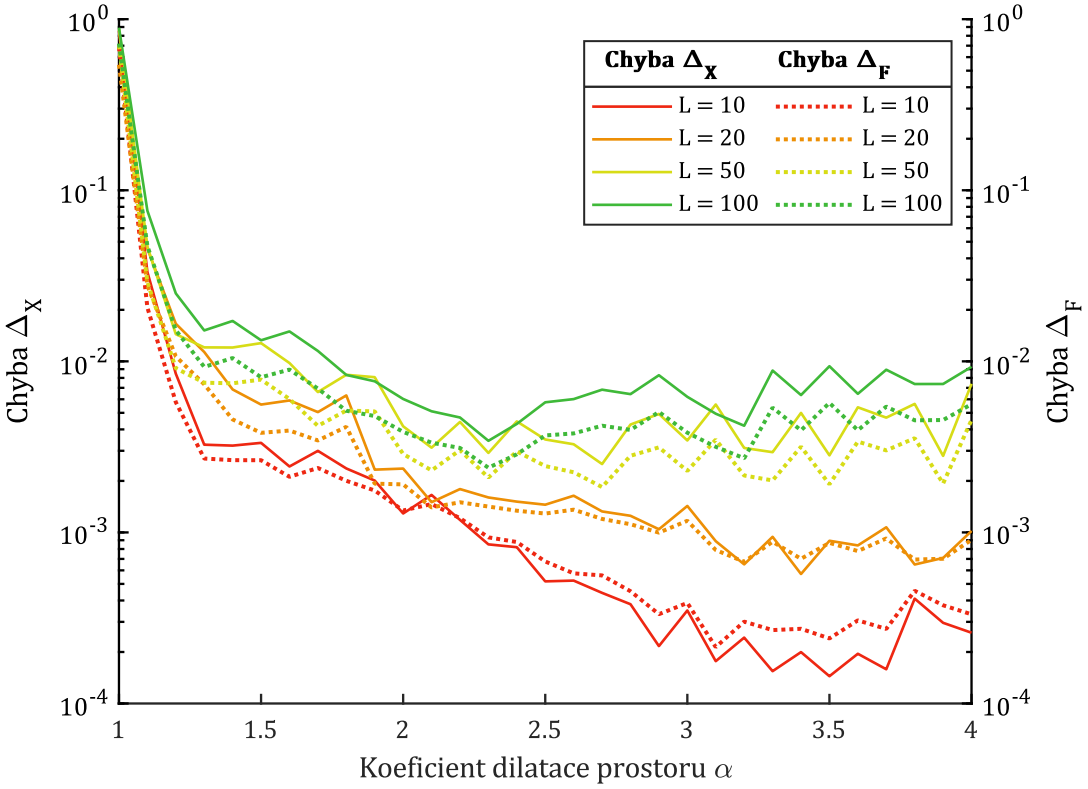
  kde Δ_X^* je chyba aproximace řešení, které není zjevně chybné (opět platí $\Delta_X \leq r_M$).

- **Rychlost výpočtu R_α** daná počtem iterací a časem (pro všechny výpočty – správné i chybné).

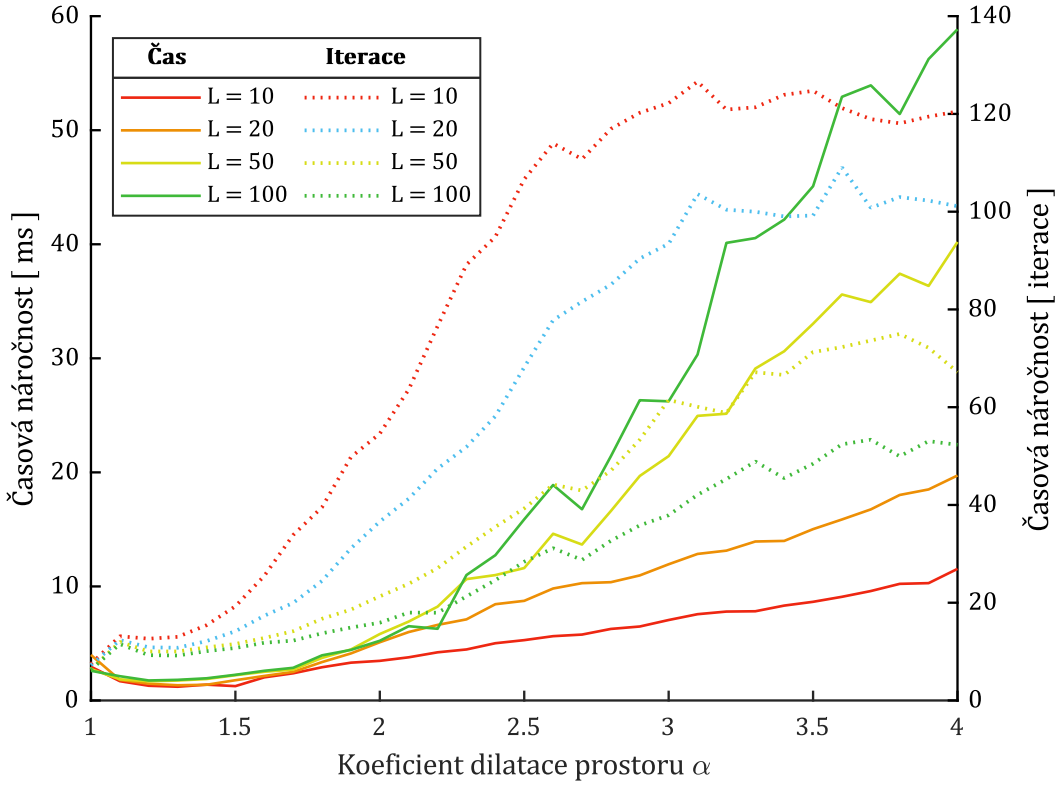
Hodnota poloměru oblasti je $r_M = 2{,}5$, počet minimalizací pro jednu hodnotu α je dán počtem uzlů sítě a tedy $C = 1271$. Kód řešení pro druhou a třetí podkapitolu uvádíme uveden v Příloze D9.



Obrázek 34– Graf závislosti spolehlivosti na koeficientu α.



Obrázek 35 – Graf závislosti přesnosti (chyby výpočtu) na koeficientu α.



Obrázek 36 – Graf závislosti výpočetní náročnosti na koeficientu α.

Parametr *L*	**10**	**20**	**50**	**100**
Označené chyby	4570 (43,9 %)	2679 (25,6 %)	1165 (11,2 %)	613 (5,90 %)
Neoznačené chyby	0	0	0	0

Tabulka 39 – Počty chybných výsledků pro jednotlivá měření.

Analýza naměřených hodnot

- **Spolehlivost, přesnost i rychlost výpočtu výrazně závisí na hodnotě *L*** – tedy jak často měníme výchozí délku kroku pro iteraci. **Vyšší hodnota *L*** znamená rychlejší a přesnější, ale současně méně spolehlivý výpočet.

- Dále učiňme závěr pro **koeficient dilatace prostoru α**. Výpočet na koeficientu α opět silně závisí.
  - Pro $L = 100$ a $L = 50$ jsme dosáhli nejmenší chyby aproximace pro $\alpha \approx 2{,}1$. Toto nastavení je současně velmi spolehlivé – přes 99 % minimalizací skončilo úspěchem – a také výpočetní čas je prakticky minimální.

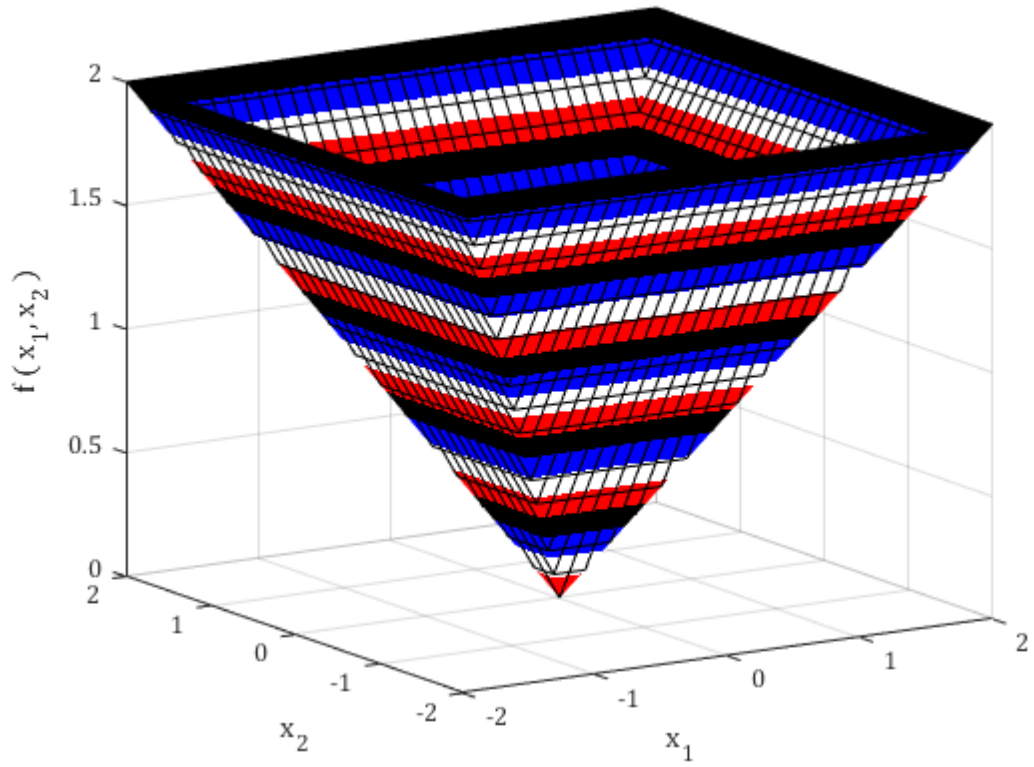
- Pro $L = 20$ a $L = 10$ chyba aproximace sice klesá až do $\alpha = 4$, avšak výpočet je pro $\alpha > 1{,}75$ nespolehlivý a také časově náročný.

- **Všechny uvedené grafy mají v praxi zcela zásadní význam**, protože uživateli sdělují, jak správně nastavit r-algoritmus a co od daného nastavení čekat. Uvedení pouze optimálních hodnot parametrů nestačí, protože každý uživatel může mít jiné preference. Uvedením grafů dáváme uživateli možnost si vybrat podle aktuálních potřeb rychlý, spolehlivý a/nebo přesný výpočet.

Závěr

Na základě měření můžeme doporučit optimální nastavení parametrů $L \in [50, 100]$ a $\alpha = 2,1$. Pro uvedené hodnoty parametrů výpočet dosahuje největší spolehlivosti i přesnosti a vysoké rychlosti.

8.4 Maxl



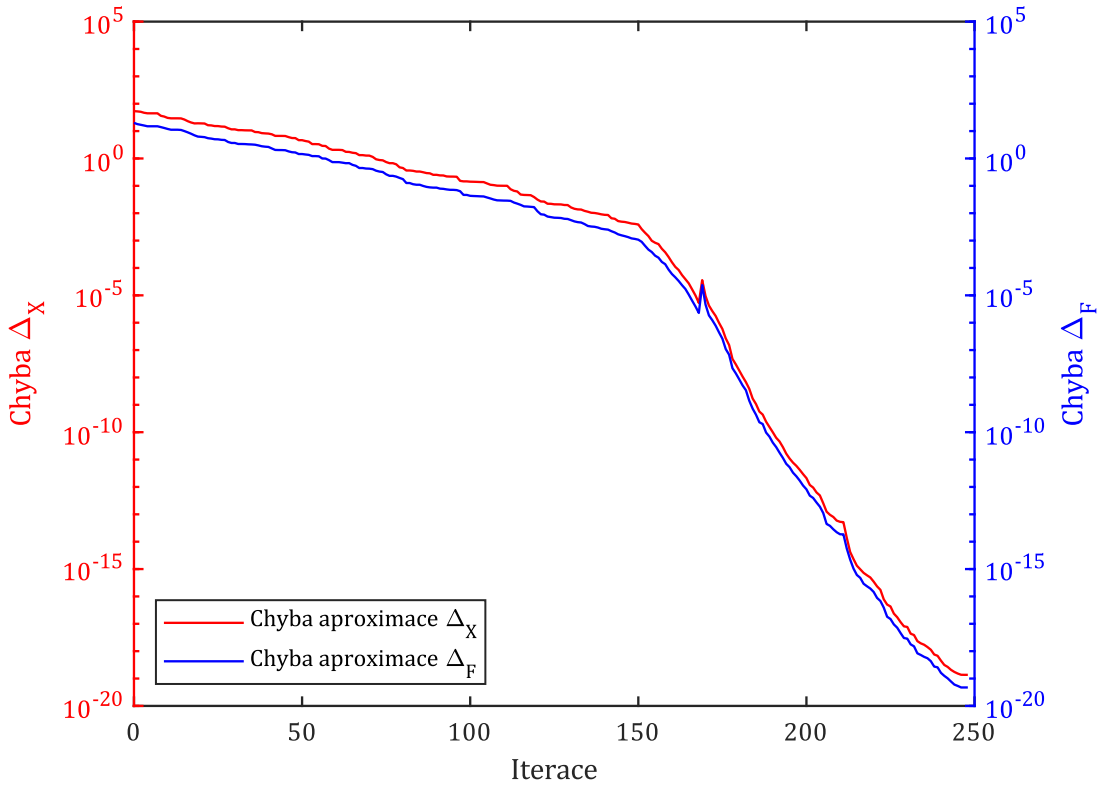
Obrázek 37 – Graf funkce Maxl pro dimenzi $d = 2$.

Pro určitou představu chování funkce Maxl vykreslíme její graf pro dimenzi $d = 2$ – *Obrázek 37*.

Funkční předpis	$f(x) = \max_{1 \le i \le d} \lvert x_i \rvert, d = 1, 2, \ldots$						
Dimenze úlohy	libovolná						
Přesné řešení	Počátek soustavy souřadnic			$f(\bar{x}) = 0$			
Počáteční aproximace	$x_0 = (1, 2, \ldots, 10, -11, -12, \ldots, -20)$ pro dimenzi $d = 20$						
Délka kroku				**Dilatace prostoru**			
h	γ	μ	L	α	p	δ	ν
0,1	0,1	1,5	Inf	1,5	10	1	10
Řízení náročnosti a běhu výpočtu							
tMax6_25	tMax7_1	Opakovani	DX	DF	StIter	Stred	
0,100 s	2,000 s	10	-1	-1	10	×	

Tabulka 40 - Zadání pro funkci Maxl.

Zvolme dimenzi $d = 20$ a určeme závislost chyby výpočtu na počtu iterací pro počáteční aproximaci $x_0 = (1, 2, \ldots, 10, -11, -12, \ldots, -20)$. Pokusem zjistíme, že minimalizace funkce vyžaduje nastavení parametru L na velmi vysokou hodnotu, ideálně nekonečno. Výsledek testu ukazuje graf na *Obrázku 38*. Ze všech zatím minimalizovaných funkcí jsme získali jednoznačně nejpřesnější výsledek. Implementaci první části úlohy najdete v Příloze D10.



Obrázek 38 – Graf závislosti chyby aproximace na počtu iterací.

8.4.1 Závislost průběhu výpočtu na dimenzi úlohy

Možnost libovolně volit dimenzi d nám dává skvělou příležitost otestovat závislost výpočtu na dimenzi úlohy. Zajímá nás počet iterací a chyba výpočtu pro dimenzi cenové funkce

$$d = 1,2,3,\dots,350.$$

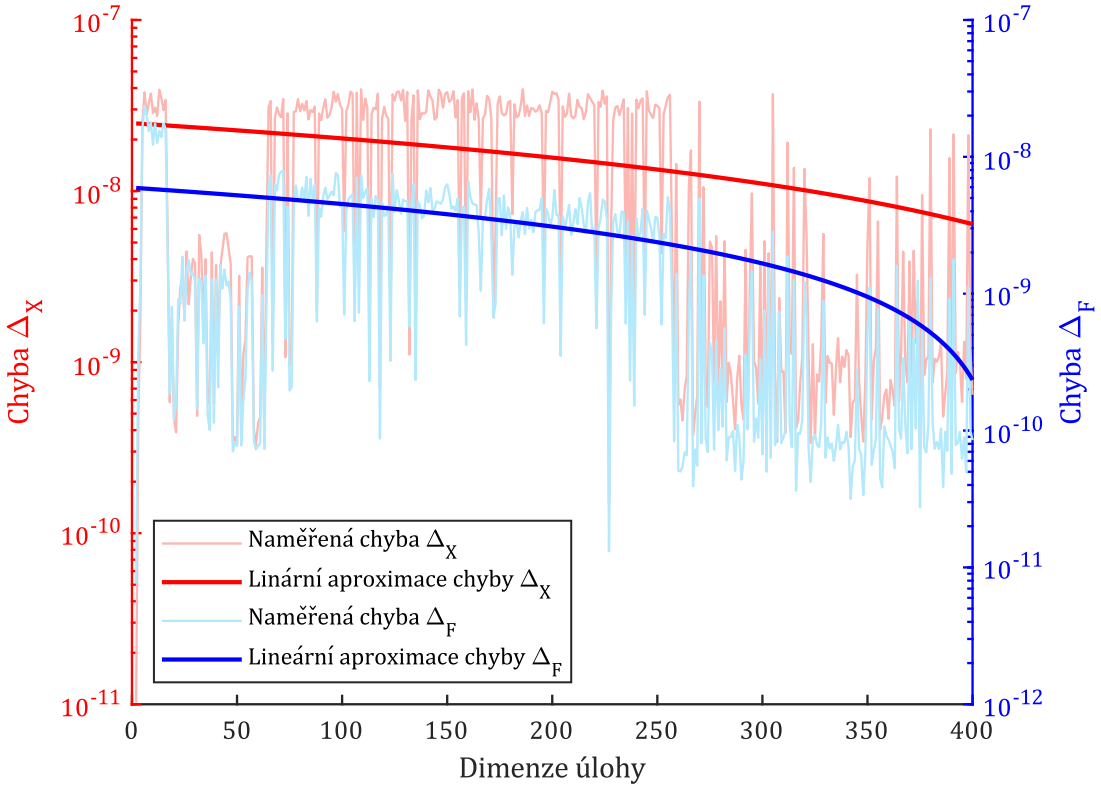
Nastavíme hodnotu parametru $L = 10$ a koeficientu $\alpha = 1{,}00$ (z důvodu řádově různých dimenzí úlohy je použití běžné hodnoty koeficientu α problematické). Implementaci najdete v Příloze D11.

Analýza výsledků

- Výpočet jsme pro všechny dimenze prováděli s konstantními parametry,
- Pro všechny dimenze jsme získali správné řešení, chyba aproximace je řádově stálá

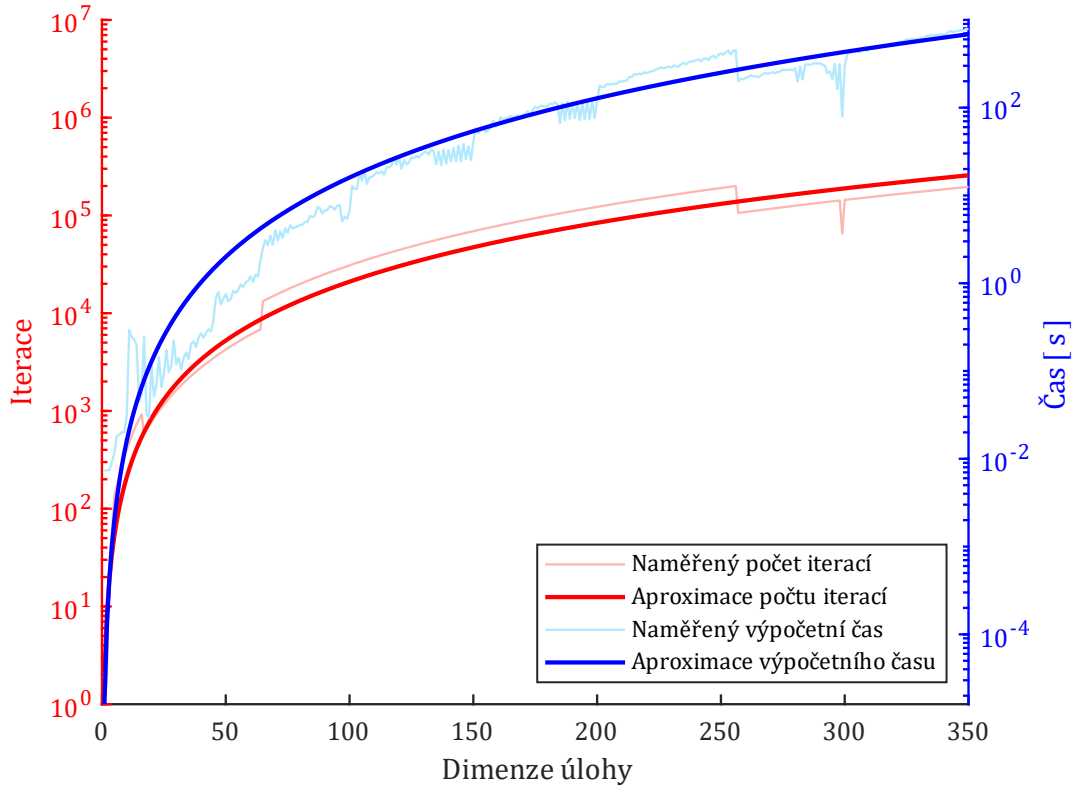
$$\bar{\Delta}_X = 1{,}6 \cdot 10^{-8} \pm 1{,}5 \cdot 10^{-8},$$
$$\bar{\Delta}_F = 2{,}6 \cdot 10^{-9} \pm 3{,}3 \cdot 10^{-9}.$$

- Počet iterací rychle roste se zvyšující se dimenzí,
- Výpočetní čas také roste. Tempo růstu je vyšší než u počtu iterací.



Obrázek 39 – Graf závislosti chyby aproximace na dimenzi úlohy – funkce Maxl.

Z grafu na *Obrázku 39* je patrné, že velikosti chyb Δ_X i Δ_F mají klesající tendenci (přesněji lineární aproximace obou chyb). My ovšem budeme obezřetní – **pro zpřesňování výpočtu u vyšších dimenzí úlohy nemáme rozumné vysvětlení.** Osobně se kloním k názoru, že daný jev způsobují zaokrouhlovací chyby. Čím vyšší dimenze úlohy, tím nižší musí být odchylky jednotlivých souřadnic aproximace bodu minima. Pro vysoké dimenze úlohy může dojít k jejich zaokrouhlení na nulu. Euklidovská norma navíc uvedený proces urychluje – při jejím výpočtu používáme kvadrát odchylky souřadnic.

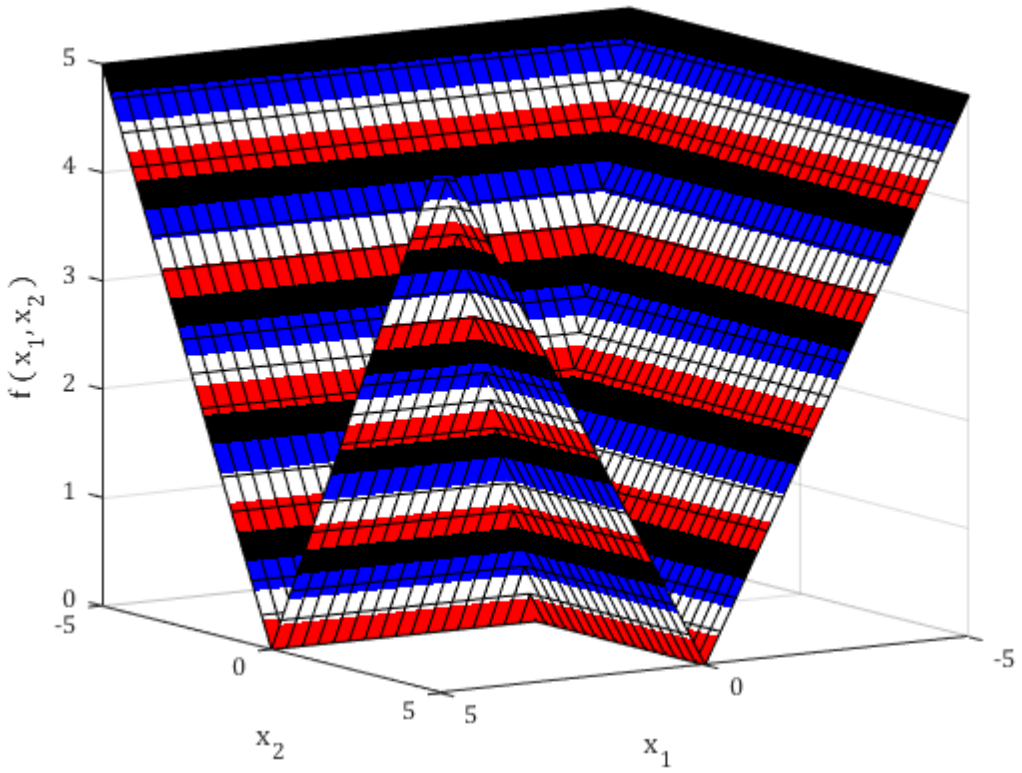


Obrázek 40 – Graf závislosti výpočetní náročnosti na dimenzi úlohy – funkce Maxl.

Nyní se přesuňme k časové náročnosti výpočtu – analyzujme graf na *Obrázku 40*. Časovou náročnost odhadneme pomocí aproximačních křivek. **Závislost počtu iterací na dimenzi úlohy je kvadratická s odhadem $\mathcal{O}(n^2)$, ale závislost výpočetního času na dimenzi úlohy je kubická s odhadem $\mathcal{O}(n^3)$.**

Obecně je sice polynomiální závislost náročnosti výpočtu přijatelná, ovšem pro výpočet v reálném čase nám nastavuje jasné mantinely pro maximální dimenzi řešené úlohy.

8.5 Neostré globální minimum



Obrázek 41 – Graf funkce s neostrým globálním minimem

Funkční předpis	$f(\boldsymbol{x}) = \left\lvert \max_{1 \le i \le 50} \{x_i\} - \sum_{i=1}^{50} x_i \right\rvert$						
Dimenze úlohy	50						
Přesné řešení	Neostré globální minimum			$f(\bar{x}) = 0$			
Počáteční aproximace	$x_i = i - 25.5, \quad i = 1,2,\dots,50$						
Délka kroku				**Dilatace prostoru**			
h	γ	μ	L	α	p	δ	ν
0,01	0,20	1,5	10	1,5	10	1	10
Řízení náročnosti a běhu výpočtu							
`tMax6_25`	`tMax7_1`	`Opakovani`	`DX`	`DF`	`StIter`	`Stred`	
0,100 s	2,000 s	10	-1	-1	10	×	

Tabulka 41 – Zadání Gofflinovy funkce.

Tato úloha se od ostatních liší – nemá ostré globální minimum. Triviálním optimem je počátek soustavy souřadnic, ale vyhovují i jiná řešení. Například body, jejichž právě jedna libovolná souřadnice je kladná a ostatní jsou nulové. Pro lepší představu o průběhu funkce vykreslíme graf její dvourozměrné varianty – *Obrázek 41*. Zadání minimalizace uvádíme v *Tabulce 41*.

Než se pustíme do samotné minimalizace, dokažme uvedené hypotézy o triviálním i netriviálním řešení. Vnější funkcí je absolutní hodnota, jejíž hodnota musí být nezáporná. Uvažujeme krajní případ

$$f(\boldsymbol{x}) = \left| \max_{1 \le i \le 50} \{x_i\} - \sum_{i=1}^{50} x_i \right| = 0.$$

Pak jsme zcela jistě našli nějaký bod minima. Jako řešení jsme uvažovali takový bod, jehož právě jedna souřadnice je **nezáporná** a ostatní jsou nulové. Označme jej obecně

$$\bar{x} = (0, \dots 0, x_k, 0, \dots, 0), x_k \geq 0.$$

Potom můžeme zápis cenové funkce zjednodušit

$$f(\boldsymbol{x}) = \left| \max_{1 \le i \le 50} \{0, \dots, 0, x_k, 0, \dots, 0\} - x_k \right| = |x_k - x_k| = |0| = 0.$$

Obě hypotézy jsou tedy pravdivé. Existují i jiná řešení? Vyjděme z dokázané skutečnosti – že existuje bod, ve kterém je funkční hodnota rovná nule (naše řešení). Pak musí pro každý jiný bod minima rovněž platit $f(x) = 0$. Úpravou cenové funkce získáme

$$f(\boldsymbol{x}) = \left| \max_{1 \le i \le 50} \{x_i\} - \sum_{i=1}^{50} x_i \right| = 0 \implies \max_{1 \le i \le 50} \{x_i\} = \sum_{i=1}^{50} x_i.$$

Množinou všech řešení (bodů minima) naší úlohy je množina

$$\overline{X} = \left\{ (x_1, x_2, \dots, x_{50}) \in \mathbb{R}^{50} \;\middle|\; \sum_{i=1}^{50} x_i = \max_{1 \le i \le 50} \{x_i\} \right\}.$$

Dále musí platit, že $\max_{1 \le i \le 50} \{x_i\} = x_k \geq 0$. Tato podmínka není typu *něco navíc*, vyplývá přímo ze zápisu řešení. Důkaz je snadný. Označme $x_k = \max_{1 \le i \le 50} \{x_i\}$, pak můžeme psát

$$x_k = \sum_{i=1}^{50} x_i = x_k + \sum_{i=1, i \neq k}^{50} x_i \,.$$

Z toho ovšem, po odečtení maximální souřadnice x_k, okamžitě plyne, že

$$\sum_{i=1, i\neq k}^{50} x_i = 0\,. \tag{8.1}$$

Nyní použijeme spor. Předpokládejme $x_k < 0$. Předpoklad dosadíme do podmínky $x_i \leq x_k$ a získáme nerovnost $x_i \leq x_k < 0$, z čehož plyne $x_i < 0$. Součet záporných čísel je ovšem záporný, což je ve sporu s (8.1). Platí tedy $x_k \geq 0$.

Nyní přejděme k **testování r-algoritmu**. Za počáteční aproximace zvolíme body uvedené v *Tabulce 42*. Ve třetím sloupci nás zajímá, jak daleko leží aproximace řešení od počátku soustavy souřadnic. Chybu Δ_X pochopitelně nemá smysl určovat. Implementaci uvádíme v Příloze D12.

Pokus	**Počáteční aproximace**	**Iterace**	**Chyba Δ_F**	**$\|\|\tilde{x}\|\|$**
1	(0,…,0)	17	$4{,}963 \cdot 10^{-6}$	0,0103
2	(1,0,…,0)	0	0	1
3	(0,…,0,3)	0	0	3
4	(-1,1,0,1,-1,1,…-1,1)	2	0,0081	7,158
5	(0,-2,0,…,0)	14	$4{,}246 \cdot 10^{-7}$	1,936
6	(1,…,1)	7	$2{,}847 \cdot 10^{-5}$	$4{,}108 \cdot 10^{-6}$
7	(1,2,…,50)	9	$2{,}177 \cdot 10^{-11}$	110,9
8	(-1,…,-25,26,…,50)	9	$1{,}497 \cdot 10^{-8}$	190,2

Tabulka 42 – Zadání a výsledky minimalizace pro funkci s neostrým globálním minimem.

Z výsledků je patrné, že počet iterací je ve všech případech nízký. Nejdéle trval výpočet pro počáteční aproximaci zvolenou v počátku, tento byl i druhý nejméně přesný. Absolutní přesnosti i rychlosti jsme dosáhli pro obě počáteční aproximace umístěné na osách, řádky 2 a 3. Vidíme, že subgradient byl roven nule a výpočet skončil v nulté iteraci. To znamená, počáteční aproximace je stejná konečná. Nejvyšší chybu výpočtu jsme naměřili pro počáteční aproximaci umístěnou do bodu minima, avšak mimo osu – řádek 4. Tato chyba byla o několik řádů vyšší než ve všech ostatních případech.

8.6 Souhrn nejdůležitějších poznatků

Provedli jsme řadu experimentů pro několik typově odlišných funkcí (zejména testovacích). Hledejme společné vlastnosti napříč všemi experimenty.

- S výjimkou kapitoly 8.4.1 byly testované cenové funkce velmi nízké dimenze, nejvýše řádu desítek. **To je pro r-algoritmus naprosto klíčová vlastnost – je vhodný pouze pro velmi nízké dimenze.** Implementace totiž obsahuje násobení matice maticí – a byť jej lze optimalizovat, nelze se jej zbavit ve smyslu snížení odhadu časové náročnosti $\mathcal{O}(n^3)$.

- **Spolehlivost výpočtu jsme zásadně zvýšili** vlastní modifikací r-algoritmu – přidáním opakovaných výpočtů v případě selhání.

- S vyšším **koeficientem dilatace prostoru α jsme dosáhli přesnějších výsledků, ale současně jsme zaznamenali i vyšší nestabilitu výpočtu**.

- Parametr L, který určuje, jak často měníme základní délku kroku, má značný vliv na délku výpočtu vyjádřenou v reálném čase. **Vyšší hodnota parametru L vede na delší, avšak relativně spolehlivý výpočet. Pro hodnotu $L \leq 20$ jsme zaznamenali velmi rychlé výpočty**, avšak někdy chybné. Jeho ideální hodnota se řádově liší pro různé funkce.

- Volba počáteční aproximace má značný vliv na správnost výpočtu v případě původní implementace r-algoritmu. Tento problém jsme odstranili jeho modifikací – přidáním časových limitů pro výpočet délky kroku a celkový výpočetní čas r-algoritmu. Současně jsme implementovali také autodetekci chybných řešení i jejich opravu. Nyní můžeme konstatovat, že **volba počáteční aproximace nemá zásadní vliv na správnost řešení**, **pokud je výpočet vhodně nastaven. Současně ovšem musíme zdůraznit, že obvykle existují počáteční aproximace, pro které výpočet selže.** Důvodem selhání je šíření numerických chyb výpočtu.

- **Vypozorovali jsme souvislost mezi vysokým počtem iterací a chybným výpočtem**, a současně se nám podařilo kvantifikovat tento počet iterací. Toho lze využít jednak pro identifikaci chybného výpočtu, a také jako další ukončovací kritérium.

- **Každá cenová funkce má jiné vlastnosti a její rychlá a spolehlivá minimalizace tudíž vyžaduje specifické nastavení výpočtu.** Nelze bezhlavě použít jakoukoli numerickou metodu, aniž bychom předem neprovedli analýzu problému. Pokud analýzu problému nemůžeme provést (složitá cenová funkce, plně automatický výpočet), použijeme obecně doporučené hodnoty všech parametrů.

9 Shrnutí práce

V této práci se zabýváme **optimalizací nehladké funkce**, důraz klademe především na koercivní cenové funkce a lokálně lipschitzovsky spojité funkce. Zabýváme se primárně implementací **r-algoritmu**, jehož autorem je Naum Zuselevich Shor. Základní variantu r-algoritmu jsme vylepšili dalšími optimalizacemi, například algoritmem pro variabilní délku kroku nebo autodetekcí chybného výpočtu včetně nápravy.

V úvodní části textu se čtenář seznámí s nezbytným teoretickým základem. Teoretickou část textu jsme doplnili o řadu ilustrací a řešených příkladů, čtenář tak snáz pochopí probíranou tematiku i její význam. Výchozím textem pro definování a pochopení nehladké optimalizace je Clarkeův kalkul, který zavádí pojem **zobecněný gradient** – rozšíření gradientu i pro body, ve kterých funkce není diferencovatelná.

Následuje seznámení se samotným r-algoritmem i dalšími, pomocnými, algoritmy – a to včetně jejich implementace v prostředí Matlabu. Poměrně obsáhlá část práce je věnována ukázkám práce algoritmu v závislosti na jeho nastavení pomocí různorodých parametrů. **Zjistili jsme, že určení optimální délky kroku je mnohem náročnější disciplína než určení směru minimalizace**.

V rámci numerických experimentů jsme testovali závislost přesnosti, rychlosti a spolehlivosti výpočtu na vstupních datech (nastavení minimalizace). Z provedených měření plyne, že **neexistuje nastavení r-algoritmu takové, které je současně rychlé, přesné i spolehlivé**. Zejména rychlost a spolehlivost často směřují proti sobě. Oproti tomu **přesnost výpočtu závisí primárně na cenové funkci**.

Zjistili jsme, že **vyšší hodnota koeficientu dilatace prostoru α vede na přesnější výpočet** (zdrojová literatura uvádí rychlejší výpočet). Nízká hodnota parametru L (znamená častější úpravu výchozí délky kroku v každé iteraci) vede na rychlejší výpočet. Kombinace nastavení pro rychlý a současně přesný výpočet (vyšší koeficient α a nižší parametr L) vede často na nespolehlivý výpočet. Z uvedeného plyne, že **nastavení jednotlivých parametrů minimalizace není nezávislé**.

Dále musíme konstatovat, že **v případě neznalosti funkční hodnoty v bodu minima je problematické stanovit i jen odhad chyby výpočtu**. Výpočet řídíme (ukončujeme) sadou maximálních chyb pro délku kroku a velikost subgradientu. Jedná se však spíše o technické prostředky k ukončení výpočtu. Snížení maximálních chyb v nastavení minimalizace nemusí nutně vést ke skutečnému snížení chyby výsledné aproximace.

Přínos této bakalářské práce

- **Vytvořili jsme modifikaci r-algoritmu**, která dokáže rozpoznat chybný výpočet už v jeho průběhu, ukončit jej, a pomocí nové počáteční aproximace provést výpočet znovu. Algoritmus tak disponuje určitou vlastní inteligenci, která **zásadním způsobem zlepšuje jeho spolehlivost** a umožňuje jeho nasazení v plně automatizovaných procesech.

- Požadavek na **časovou stabilitu výpočtu** jsme vyřešili zavedením časových limitů pro stanovení optimální délky kroku a běhu celého r-algoritmu. **Uživatel může – podle svých priorit – nastavit maximální dobu výpočtu, což umožňuje nasazení implementace pro výpočty v reálném čase**. Oproti ukončení výpočtu z příkazu nadřazeného programu je naše řešení výhodnější – k ukončení výpočtu dochází na vhodném místě programu – vždy po skončení celé iterace. V rámci experimentů jsme navíc ukázali, že uvedené omezení má na přesnost výpočtu pouze minimální až zanedbatelný vliv, protože dlouhé výpočty (časově i co do počtu iterací) obvykle vedou na chybný výsledek.

- **Měřením jsme určili závislost charakteristik výpočtu na nastavení minimalizace pro široké spektrum řídících parametrů**. Z grafů i tabulek čtenář snadno vyčte, jaké nastavení minimalizace může použít a také co od tohoto nastavení čekat.

10 Použitá literatura

[1] DOSTÁL, Zdeněk a Petr BEREMLIJSKI. *Metody optimalizace*. Přepracované vydání. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2018.

[2] DUPAČOVÁ, Jitka a Petr LACHOUT. *Úvod do optimalizace*. Přepracované vydání. Praha: Matfyzpress, 2011. ISBN 978-80-7378-176-7.

[3] FEISTAUER, Miloslav a Václav KUČERA. *Základy numerické matematiky*. Přepracované vydání. Praha: Matfyzpress, 2014. ISBN 978-80-7378-264-1.

[4] STANOVSKÝ, David a Libor BARTO. *Počítačová algebra*. Druhé, upravené vydání. Praha: Matfyzpress, 2017. ISBN 978-80-7378-340-2.

[5] KARMITSA, N., A. BAGIROV a M. M. MÄKELÄ (2012): Comparing different nonsmooth minimization methods and software, Optimization Methods and Software, 27:1, 131-153

[6] VONDRÁK, Vít a Lukáš POSPÍŠIL. *Numerické metody I*. Ostrava/Plzeň: Vysoká škola báňská – Technická univerzita Ostrava a Západočeská univerzita v Plzni, 2011.

[7] MUSILOVÁ, Jana a Pavla MUSILOVÁ. *Matematika pro porozumění i praxi: netradiční výklad tradičních témat vysokoškolské matematiky*. Brno: VUTIUM, 2017. ISBN 978-80-214-5503-0.

[8] SAWA, Zdeněk. *Úvod do teoretické informatiky: Logika a algoritmy*. Pracovní verze. Ostrava: VŠB - Technická univerzita, 2014.

[9] SHOR, N.Z., K.C. KIWIEL a A. RUSZCZYŃSKI. *Minimization Methods for Non-Differentiable Functions*. Softcover reprint of the hardcover 1st edition 1985. Heidelberg: Springer-Verlag Berlin, 1985. ISBN 978-3-642-82120-2.

[10] KUBEN, Jaromír a Petra ŠARMANOVÁ. *Diferenciální počet funkcí jedné proměnné*. Ostrava: VŠB - Technická univerzita, 2006. ISBN 80-248-1192-8.

[11] KUBEN, Jaromír, Š. MAYEROVÁ, P. RAČKOVÁ a Petra ŠARMANOVÁ. *Diferenciální počet funkcí více proměnných*. Ostrava: VŠB - Technická univerzita, 2012.

11 Seznam příloh

A Referenční počítač 109

B Použité optimalizační technologie 110

C Implementace probraných algoritmů 111

C.1 Numerický výpočet gradientu 111

C.2 Výpočet nové aproximace pro délku kroku danou předem 112

C.3 Výpočet nové aproximace pro variabilní délku kroku 113

C.4 Implementace metody největšího spádu – NS 114

C.5 Optimalizace matice B 116

C.6 Implementace metody SDG 117

C.7 Shorův r-algoritmus 119

C.8 Měření časové náročnosti výpočtu 121

C.9 Výpočet nové aproximace (modifikace) 124

C.10 Ověření aproximace na platnost zadaných kritérií 125

C.11 Shorův r-algoritmus (modifikace) 126

D Zdrojový kód řešených úloh 130

D.1 Příklad 6.32 130

D.2 Příklad 6.56 131

D.3 Příklad 6.57 132

D.4 Příklad 7.3 133

D.5 Rosenbrock – 1. část 134

D.6 Rosenbrock – 2. část 136

D.7 Půlměsíc 138

D.8 Charalambous-Bandler – 1. část 140

D.9 Charalambous-Bandler – 2. část 141

D.10 Maxl – Dimenze 20 143

D.11 Maxl – Závislost výpočtu na dimenzi úlohy 144

D.12 Neostré globální minimum 145

E Elektronické přílohy – Informační systém Edison .

E.1 Pokračování numerických experimentů

E.2 Implementace metod a úloh pro prostředí Matlab .

A. Referenční počítač

V některých úloh jsme uváděli časovou náročnost v reálném čase. Ta má pro některé algoritmy vyšší vypovídající hodnotu, avšak je závislá na výkonnosti použitého počítače. Pro výpočty v této práce jsme použili high-end domácí počítač s následující konfigurací.

Název	**Počet**	**Typ**	**Popis**
Procesor	1	Intel Core i9-10940X	14 jader, 28 vláken, 3,30 GHz, TurboBoost 4,8 GHz, TDP 165 W, reálná spotřeba až 350* W.
Paměť RAM	8	Corsair Vengeance 32 GB DDR4	256 GB, 3200 MHz, XMP 2.0
Grafická karta	2	NVIDIA GeForce RTX 2080 Ti	4352 CUDA jader, 1545 MHz, 11 GB RAM DDR6 (616 GB/s), NVLink Bridge, spotřeba 250 W.
Základní deska	1	MSI Creator X299	Podpora Quad-Channel DDR4
Paměť	2	Samsung 970 EVO Plus 2 TB	V-NAND SSD NVMe M.2, Sekvenční čtení až 3,5 GB/s, Sekvenční zápis až 3,2 GB/s.
Chlazení CPU	1	Corsair H150i RGB PRO XT	Vodní, 3x120mm, až 350 W.
Zdroj	1	Corsair HXi Series HX1200i	Výkon 1200 W.
OS		Microsoft Windows 10 Pro	

Krátce se zastavme u procesoru, jehož architektura je Cascade Lake-X (Enthusiast). Tato architektura vznikla optimalizací architektury Sky Lake-X. Jedná se o extrémně výkonný škálovatelný procesor určený pro fanoušky výpočetní techniky, svými vlastnostmi leží někde mezi standardními a serverovými procesory (architekturu přebral od serverových). Pomocí základní desky jej lze taktovat i na frekvence vyšší než 4,8 GHz, nicméně při 14 jádrech bychom procesor nezvládli uchladit – spotřeba energie roste exponenciálně. Při 14 jádrech a frekvenci 5,0 GHz by výkon dosahoval zhruba 750 Wattů.

Co se velikosti paměti RAM týče, musí dbát na její efektivní správu. Přenos 200 GB dat na pevný disk trvá při průměrné rychlosti zápisu 2 GB/s celých 100 sekund. Obecně je potřeba minimalizovat přenos dat mezi pevným diskem, pamětí RAM i pamětí CPU.

B. Použité optimalizační technologie

Za zmínku určitě stojí technologie umožňující optimalizaci výpočtu - paralelní zpracování dat na CPU. Touto technologií je **BLAS (*Basic Linear Algrebra Subroutins)***. V našem případě – pro procesory Intel – se jedná o součást knihovny **MKL (*Intel Math Kernel Library*)** od společnosti Intel. Tuto knihovnu Matlab využívá pro násobení vektorů a matic. Uvedené operace totiž nejsou součástí kódu Matlabu.

Existují tři úrovně výpočtů optimalizovaných pomocí BLAS:

1. Násobení vektoru vektorem,
2. Násobení vektoru maticí,
3. Násobení matice maticí.

Procesory společnosti AMD používaly dříve ACML (AMD Core Math Library), avšak v současnosti AMD již vlastní optimalizaci nevyvíjí a uživatelé těchto procesorů jsou odkázáni na volně šiřitelnou platformu OpenBLAS (výkonnost této optimalizace v Matlabu je předmětem diskusí).

Při numerických experimentech jsme používali **paralelní cyklus `parfor`**. Jeho použití má sice řadu omezení, ale umožňuje výpočet na libovolném počtu jader současně (počet lze nastavit v Matlabu). Toto řešení je přímo součástí Matlabu a dobře spolupracuje s BLASem. Nevýhodou této spolupráce je nemožnost dopředu určit, jak bude výpočet proveden. Technologie BLAS je sice volně šiřitelná, ale není otevřená – neznáme její implementaci.

Nakonec zmiňme technologii **Hyper-Threading (HT, HTT)** od společnosti Intel. Technologie vytváří z jednoho fyzického jádra dvě virtuální. Uvedená optimalizace je mimořádně účinná pro matematické operace. BLAS technologii HT plně využívá, zatímco cyklus `parfor` pracuje ve výchozím nastavení pouze s fyzickými jádry.

C. Implementace probraných algoritmů

Abychom se v kódech vyznali, v Matlabu nejdříve založíme projekt, například s názvem Bakalářka. Volíme spíše programování funkcí pro každý algoritmus. Některé postranní úlohy mohou být řešené skriptem, obvykle srovnávací testy, vykreslení konkrétního grafu a podobně. Programujeme s ohledem na maximální efektivitu výpočtu. Funkce a skripty vhodně pojmenujeme a rozumně komentujeme. Robustnost funkcí neuvažujeme, předpokládáme formálně bezchybné zadání vstupů.

1. Numerický výpočet gradientu

Funkce rovněž slouží pro výpočet zobecněného gradientu, v tomto případě počítá střední hodnotu každé složky gradientu. Použit je lineární model.

```
Soubor NumGrad.m

function [g] = NumGrad(f, x0, zeta)

%NUMGRAD Numerický výpočet gradientu
%
%   Pro hladkou i nehladkou funkci
%
%   [g] = NumGrad(f, x0, zeta)
%
%f   Spojitá funkce v bodě x0
%x0   Bod, ve kterém gradient počítáme
%zeta   Poloměr okolí, pomocí kterého počítáme gradient
%g   Gradient - sloupcový vektor

n = length(x0);
g = zeros(n,1);
E = eye(n);

for i = 1:n
    g(i) = (1/(2*zeta)) * ( f(x0 + zeta*E(:,i)) - f(x0 - zeta*E(:,i)) );
end

end
```

Zdrojový kód: Numerický výpočet gradientu

2. Výpočet nové aproximace pro délku kroku danou předem

Před zahájením iteračního výpočtu je metodu nutné nejdříve nastavit (postup uvádíme v popisu funkce).

```
Soubor AproxApriori.m

function [naprox, krok] = AproxApriori(typ, data)

%APROXAPRIORI Výpočet nové aproximace pro délku kroku danou předem
%
%    [naprox, krok] = AproxApriori(typ, data)
%
%typ    'set' -> Nastavení metody
%       data = {c, x0}
%       c = poloměr okolí, x0 = počáteční aproximace
%       Poznámka: Před iteračním výpočtem aproximací je nutné
%       nejdříve nastavit metodu podle tohoto odstavce
%
%typ    'cokoliv jiného' -> Výpočet nové aproximace
%       data = {eta}
%       eta = Jednotkový směr poklesu
%
%naprox    Nová aproximace
%krok    Délka kroku

persistent c aprox k;

switch typ
    case 'set'
        k = 0;
        c = data{1};
        aprox = data{2};
        krok = c;

    otherwise
        krok = c / (k + 1);
        aprox = aprox + data{1} * krok;
        k = k + 1;
end

naprox = aprox;

end
```

Zdrojový kód: Výpočet nové aproximace pro konvexní funkce

3. Výpočet nové aproximace pro variabilní délku kroku

Před zahájením iteračního výpočtu je metodu nutné nejdříve nastavit (postup uvádíme v popisu funkce).

Soubor AproxVariabilni.m

```
function [naprox, krok] = AproxVariabilni(typ, data)

%APROXVARIABILNI Výpočet nové aproximace pro variabilní délku kroku
%
%   [naprox, krok] = AproxVariabilni(typ, data)
%
%typ   'set' -> Nastavení metody
%      data = {f, x0, h, gamma, my, L}
%      f   Cenová funkce
%      x0   Počáteční aproximace
%      h   Základní délka kroku
%      gamma   Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%      my   Koeficient my >= 1 pro zvětšení délky kroku
%      L   Počet zvětšení kroku do zvětšení jeho základní délky
%      Poznámka: Před iteračním výpočtem aproximací je nutné
%      nejdříve nastavit metodu podle tohoto odstavce
%
%typ   'cokoliv jiného' -> Výpočet nové aproximace
%      data = {eta}
%      eta = Transformovaný směr poklesu
%
%naprox   Nová aproximace
%krok   Délka kroku

persistent f aprox h gamma my L

switch typ
    case 'set'
        f = data{1};
        aprox = data{2};
        h = data{3};
        gamma = data{4};
        my = data{5};
        L = data{6};

    otherwise
        t = aprox + h * data{1};

        if f(t) >= f(aprox)
            aprox = t;
            h = gamma * h;

        else
            k = 0;

            tic;
            while f(t) < f(aprox)
```

```
                CasBehu = toc;
                if CasBehu > 5
                    break;
                end

                aprox = t;
                t = t + h * data{1};
                k = k + 1;
            end

            if k > L
            h = k * my * h / L;
            end

        end
end

naprox = aprox;
krok = h;

end
```

Zdrojový kód: Výpočet nové aproximace pro koercivní funkce

4. Implementace metody největšího spádu – NS

```
Soubor NejSpad.m

function [xmin, iter, kod] = NejSpad(Funkce, Metoda, Data)

%NEJSPAD Metoda největšího spádu pro nehladkou funkci
%
%   [xmin, iter, kod] = NejSpad(Funkce, Metoda, Data)
%
%Funkce   Struktura {
%         f; cenová funkce
%         x0; počáteční aproximace
%         zeta; Poloměr okolí pro výpočet numerického subgradientu
%         fmin; Hodnota minima (volitelný parametr)
%         }
%
%Metoda   'apriori' -> Délka kroku daná předem bez SD
%         Data = struktura {
%         c; Průměr množiny možných řešení
%         e; Vektor přesnosti
%                e = (e1, e2, e3, e4, e5)
%                --- Zadat požadované hodnoty ---
%                e1 = velikost subgradientu
%                e2 = velikost transformovaného subgradientu
%                e3 = délka kroku
%                e4 = vzdálenost dvou po sobě jdoucích aproximací
```

```
%                       || x(k+1) - x(k) ||
%                   e5 = Velikost rozdílu hodnot minima aproximace minima
%                       | fmin - f(x(k)) |
%                   Poznámka: Pokud některé kritérium nepoužijeme,
%                   pak jeho hodnota musí být nastavena na -1
%           }
%
%Metoda     'variabilni' -> Variabilní délka kroku bez SD
%           Data = struktura {
%           h; Základní délka kroku
%           gamma; Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%           my; Koeficient my >= 1 pro zvětšení délky kroku
%           L; Počet zvětšení kroku do zvětšení jeho základní délky
%           e; Vektor přesnosti stejně jako u konvexní varianty
%
%xmin       Argument minima cenové funkce
%iter       Počet provedených iterací
%kod        Kód zastavovací podmínky podle vektoru přesnosti e


xmin = Funkce{2};
iter = 0;

switch Metoda
    case 'apriori'
        AproxApriori('set', {Data{1}, Funkce{2}});
        Presnost = Data{2};
    case 'variabilni'
        F = {Funkce{1}, Funkce{2}, Data{1}, Data{2}, Data{3}, Data{4}};
        AproxVariabilni('set', F);
        Presnost = Data{5};
end

while 1
    xmin2 = xmin;
    g = NumGrad(Funkce{1}, xmin, Funkce{3});
    gn = norm(g);
    if Presnost(1) > gn
        kod = 1;
        break;
    end

    g = g/gn;
    iter = iter + 1;

    switch Metoda

        case 'apriori'
            [xmin, krok] = AproxApriori('provoz', {-1*g});

        case 'variabilni'
            [xmin, krok] = AproxVariabilni('provoz', {-1*g});
            dx = norm(xmin2 - xmin);
            if Presnost(4) > dx
                kod = 4;
                break;
```

```
            end
    end

    if Presnost(3) > krok
        kod = 3;
        break;
    end

    if Presnost(5) ~= -1
        dfx = abs( Funkce{1}(xmin2) - Funkce{1}(xmin) );
        if Presnost(5) > dfx
            kod = 5;
            break;
        end
    end

end

end
```

Zdrojový kód: Implementace metody největšího spádu

5. Optimalizace matice *B*

```
Soubor OptiB.m

function [B] = OptiB(B, delta, ny)

%OPTIB Optimalizace matice B
%
%   [B] = OptiB(B, delta, ny)
%
%delta   Nejmenší přípustná hodnota největšího prvku |B|
%ny    Koeficient ny > 1 pro násobení matice B
%
%B   Matice B (vstupní i výstupní parametr)

d = max(max(abs(B)));
theta = delta / d;

if theta > 1

    B = ny * theta * B;

end

end
```

Zdrojový kód: Optimalizace matice B

6. Implementace metody SDG

```
Soubor SDG.m

function [xmin, iter, kod] = SDG(Funkce, Krok, SD)

%SDG Metoda s dilatací prostoru ve směru gradientu
%
%   [xmin, iter, kod] = SDG(Funkce, Krok, SD)
%
%Funkce   Struktura {
%         f; cenová funkce
%         x0; počáteční aproximace
%         zeta; Poloměr okolí pro výpočet numerického subgradientu
%         fmin; Hodnota minima (volitelný parametr)
%         }
%
%Krok     Struktura {
%         h; Základní délka kroku
%         gamma; Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%         my; Koeficient my >= 1 pro zvětšení délky kroku
%         L; Počet zvětšení kroku do zvětšení jeho základní délky
%         e; Vektor přesnosti
%              e = (e1, e2, e3, e4, e5)
%              --- Zadat požadované hodnoty ---
%              e1 = velikost subgradientu
%              e2 = velikost transformovaného subgradientu
%              e3 = délka kroku
%              e4 = vzdálenost dvou po sobě jdoucích aproximací
%                   || x(k+1) - x(k) ||
%              e5 = Velikost rozdílu hodnot minima aproximace minima
%                   | fmin - f(x(k)) |
%              Poznámka: Pokud některé kritérium nepoužijeme,
%              pak jeho hodnota musí být nastavena na -1
%         }
%
%SD       {
%         alfa; Koeficient dilatace prostoru
%         p; Počet iterací - interval optimalizace matice B
%         delta; Nejmenší přípustná hodnota největšího prvku |B|
%         ny; Koeficient ny > 1 pro násobení matice B
%         }
%
%xmin   Argument minima cenové funkce
%iter   Počet provedených iterací
%kod   Kód zastavovací podmínky podle vektoru chyb e

xmin = Funkce{2};
n = length(xmin);
I = eye(n);
B = I;
kappa = 1/SD{1} - 1;
iter = 0;

F = {Funkce{1}, Funkce{2}, Krok{1}, Krok{2}, Krok{3}, Krok{4}};
AproxVariabilni('set', F);
```

```
Presnost = Krok{5};

while 1
    xmin2 = xmin;
    g = NumGrad(Funkce{1}, xmin, Funkce{3});
    if Presnost(1) > norm(g)
        kod = 1;
        break;
    end

    gh = B' * g;
    ghn = norm(gh);
    if ghn < Presnost(2)
        kod = 2;
        break;
    end

    ksi = gh/ghn;
    eta = -B * ksi;
     iter = iter + 1;

    [xmin, krok] = AproxVariabilni('provoz', {eta});
    dx = norm(xmin2 - xmin);
    if Presnost(4) > dx
        kod = 4;
        break;
    end

    if Presnost(3) > krok
        kod = 3;
        break;
    end

    if Presnost(5) ~= -1
        dfx = abs( Funkce{1}(xmin2) - Funkce{1}(xmin) );
        if Presnost(5) > dfx
            kod = 5;
            break;
        end
    end

    B = B * ( I + (kappa * (ksi * ksi') ) );

    if mod(iter,SD{2}) == 0
        B = OptiB(B, SD{3}, SD{4});
    end

end

end
```

Zdrojový kód: Implementace SDG metody

7. Shorův r-algoritmus

```
Soubor Shor.m

function [xmin, iter, kod] = Shor(Funkce, Krok, SD)

%SHOR Shorův r-algoritmus
%
%   [xmin, iter, kod] = Shor(Funkce, Krok, SD)
%
%Funkce   Struktura {
%         f; cenová funkce
%         x0; počáteční aproximace
%         zeta; Poloměr okolí pro výpočet numerického subgradientu
%         fmin; Hodnota minima (volitelný parametr)
%         }
%
%Krok     Struktura {
%         h; Základní délka kroku
%         gamma; Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%         my; Koeficient my >= 1 pro zvětšení délky kroku
%         L; Počet zvětšení kroku do zvětšení jeho základní délky
%         e; Vektor přesnosti
%              e = (e1, e2, e3, e4, e5)
%              --- Zadat požadované hodnoty ---
%              e1 = velikost subgradientu
%              e2 = velikost transformovaného subgradientu
%              e3 = délka kroku
%              e4 = vzdálenost dvou po sobě jdoucích aproximací
%                   || x(k+1) - x(k) ||
%              e5 = Velikost rozdílu hodnot minima aproximace minima
%                   | fmin - f(x(k)) |
%              Poznámka: Pokud některé kritérium nepoužijeme,
%              pak jeho hodnota musí být nastavena na -1
%         }
%
%SD       {
%         alfa; Koeficient dilatace prostoru
%         p; Počet iterací - interval optimalizace matice B
%         delta; Nejmenší přípustná hodnota největšího prvku |B|
%         ny; Koeficient ny > 1 pro násobení matice B
%         }
%
%xmin   Argument minima cenové funkce
%iter   Počet provedených iterací
%kod   Kód zastavovací podmínky podle vektoru chyb e

xmin = Funkce{2};
n = length(xmin);
I = eye(n);
B = I;
kappa = 1/SD{1} - 1;
iter = 0;
Presnost = Krok{5};

%pro experimentální účely
```

```
casMax = 10;
tic;

gv = NumGrad(Funkce{1}, xmin, Funkce{3});
if norm(gv) < Presnost(2)
    kod = 2;
    return;
end

F = {Funkce{1}, Funkce{2}, Krok{1}, Krok{2}, Krok{3}, Krok{4}};
AproxVariabilni('set', F);

eta = - gv / norm(gv);
[xmin, ~] = AproxVariabilni('provoz', {eta});

while 1
    xmin2 = xmin;
    g = NumGrad(Funkce{1}, xmin, Funkce{3});
    if Presnost(1) > norm(g)
        kod = 1;
        break;
    end

    gh = B' * g;
    if norm(gh) < Presnost(2)
        kod = 2;
        break;
    end

    r = gh - gv;
    rn = norm(r);
    if rn < Presnost(2)
        kod = 2;
        break;
    end
    ksi = r/rn;

    B = B * ( I + (kappa * (ksi * ksi') ) );

    if mod(iter,SD{2}) == 0 && iter ~= 0
        B = OptiB(B, SD{3}, SD{4});
    end

    gv = B' * g;
    if norm(gv) < Presnost(2)
        kod = 2;
        break;
    end

    eta = -B * gv;
    iter = iter + 1;

    [xmin, krok] = AproxVariabilni('provoz', {eta});
    dx = norm(xmin2 - xmin);
    if Presnost(4) > dx
```

```
        kod = 4;
        break;
    end

    if Presnost(3) > krok
        kod = 3;
        break;
    end

    if Presnost(5) ~= -1
        dfx = abs( Funkce{1}(xmin2) - Funkce{1}(xmin) );
        if Presnost(5) > dfx
            kod = 5;
            break;
        end
    end

    %pro experimentální účely
    casAkt = toc;
    if casAkt > casMax
        kod = 10;
        break;
    end

end

end
```

Zdrojový kód: Základní varianta r-algoritmu

8. Měření časové náročnosti výpočtu

```
Soubor Experimenty/CasovaNarocnost.m

function [Vysledky, iter, kod] = CasovaNarocnost(Funkce, Krok, ...
    SD, MAXiter)

%CASOVANAROCNOST Závislost přesnosti na počtu iterací
%
%   Pro minimalizaci algoritmem r-Shor
%
%   [xmin, iter, kod] = CasovaNarocnost(Funkce, Krok, SD, iterace)
%
%Funkce   Struktura {
%         f; cenová funkce
%         x0; počáteční aproximace
%         zeta; Poloměr okolí pro výpočet numerického subgradientu
%         xmin; Skutečný bod minima
%         }
%
%Krok     Struktura {
%         h; Základní délka kroku
```

```
%         gamma; Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%         my; Koeficient my >= 1 pro zvětšení délky kroku
%         L; Počet zvětšení kroku do zvětšení jeho základní délky
%         }
%
%SD       {
%         alfa; Koeficient dilatace prostoru
%         p; Počet iterací - interval optimalizace matice B
%         delta; Nejmenší přípustná hodnota největšího prvku |B|
%         ny; Koeficient ny > 1 pro násobení matice B
%         }
%
%MAXiter  Maximální počet iterací, výpočet se poté ukončí
%         bez ohledu na vše ostatní
%
%Vysledky   Výsledky měření {iter, vlastnosti B, x, max{dx,df}}
%iter   Počet provedených iterací
%kod   Kód zastavovací podmínky:
%           0 = proběhl maximální počet iterací
%           1 = nulová velikost subgradientu
%           2 = nulová velikost transformace subgradientu
%           3 = nulová velikost kroku

xReseni = Funkce{4};
fReseni = Funkce{1}(xReseni);
xmin = Funkce{2};

Vysledky = cell(4, MAXiter);
Vysledky(:,1) = {0; xReseni; norm(xmin-xReseni); ...
    abs(fReseni-Funkce{1}(xmin))};

n = length(xmin);
I = eye(n);
B = I;
kappa = 1/SD{1} - 1;
kod = 0;

gv = NumGrad(Funkce{1}, xmin, Funkce{3});
if norm(gv) == 0
    kod = 1;
    return;
end

F = {Funkce{1}, Funkce{2}, Krok{1}, Krok{2}, Krok{3}, Krok{4}};
AproxVariabilni('set', F);

eta = - gv / norm(gv);
[xmin, ~] = AproxVariabilni('provoz', {eta});

for iter = 2 : MAXiter
    xmin2 = xmin;
    g = NumGrad(Funkce{1}, xmin, Funkce{3});
    if norm(g) == 0
        kod = 1;
        break;
    end
```

```
    gh = B' * g;
    if norm(gh) == 0
        kod = 2;
        break;
    end

    r = gh - gv;
    rn = norm(r);
    if rn == 0
        kod = 2;
        break;
    end
    ksi = r/rn;

    B = B * ( I + (kappa * (ksi * ksi') ) );

    if mod(iter,SD{2}) == 0 && iter ~= 0
        B = OptiB(B, SD{3}, SD{4});
    end

    gv = B' * g;
    if norm(gv) == 0
        kod = 2;
        break;
    end

    eta = -B * gv;

    [xmin,~] = AproxVariabilni('provoz', {eta});

    %vložit údaj do statistiky
    Vysledky(:,iter) = {iter-1; xmin; norm(xmin-xReseni); ...
        abs(Funkce{1}(xmin)-fReseni)};

    dx = norm(xmin2 - xmin);
    if dx == 0
        kod = 3;
        break;
    end

end

if iter < MAXiter
    if (kod==1) && (kod==2)
        iter = iter - 1;
    end
    Vysledky2 = Vysledky;
    Vysledky = cell(4,iter);
    Vysledky(:,:) = Vysledky2(:,1:iter);
end

end
```

Zdrojový kód: Zjištění časové náročnosti a maximální přesnosti výpočtu

9. Výpočet nové aproximace (modifikace)

Soubor AproxVariabilni2.m

```
function [naprox, krok] = AproxVariabilni2(typ, data)

%APROXVARIABILNI2 Výpočet nové aproximace při minimalizaci funkce
%
%   [naprox, krok] = AproxVariabilni2(typ, data)
%
%typ   'set' -> Nastavení metody
%      data = {f, x0, h, gamma, my, L, tMax}
%      f   Cenová funkce
%      x0   Počáteční aproximace
%      h   Základní délka kroku
%      gamma   Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%      my   Koeficient my >= 1 pro zvětšení délky kroku
%      L   Počet zvětšení kroku do zvětšení jeho základní délky
%      tMax   Maximální čas běhu cyklu while
%      Poznámka: Před iteračním výpočtem aproximací je nutné
%      nejdříve nastavit metodu podle tohoto odstavce
%
%typ   'cokoliv jiného' -> Výpočet nové aproximace
%      data = {eta}
%      eta = Transformovaný směr poklesu
%
%naprox   Nová aproximace
%krok   Délka kroku

persistent f aprox h gamma my L tMax

switch typ
    case 'set'
        f = data{1};
        aprox = data{2};
        h = data{3};
        gamma = data{4};
        my = data{5};
        L = data{6};
        tMax = data{7};

    otherwise
        t = aprox + h * data{1};

        if f(t) >= f(aprox)
            aprox = t;
            h = gamma * h;

        else
            k = 0;

            tic;
            while f(t) < f(aprox)
```

```
                if toc > tMax
                    break;
                end

                aprox = t;
                t = t + h * data{1};
                k = k + 1;
            end

            if k > L
            h = k * my * h / L;
            end

        end
end

naprox = aprox;
krok = h;

end
```

Zdrojový kód: Výpočet nové aproximace pro koercivní funkce (modifikace)

10. Ověření aproximace na platnost zadaných kritérií

```
Soubor OverVysledek.m

function Overeni = OverVysledek(xmin, Funkce, Rizeni)

%OVERVYSLEDEK Ověří aproximaci na platnost kritérií DX a DF
%
%   Overeni = OverVysledek(xmin, Funkce, Rizeni)
%
%   Význam vstupních parametrů je shodný s metodou Shor2
%
%Overeni    0 = Výsledek nesplňuje limity
%           1 = Výsledek splňuje limity

Overeni = 1;

if Rizeni{4} ~= -1 %Ověření limitu pro DX
    dx = norm(xmin - Rizeni{7});
    if dx > Rizeni{4}
        Overeni = 0;
        return;
    end
end

if Rizeni{5} ~= -1 %Ověření limitu pro DF
    df = abs(Funkce{4} - Funkce{1}(xmin));
    if df > Rizeni{5}
```

```
        Overeni = 0;
        return;
    end
end

end
```

Zdrojový kód: Ověření aproximace na platnost zadaných kritérií

11. Shorův r-algoritmus (modifikace)

```
Soubor Shor2.m

function [xmin, iter, kod] = Shor2(Funkce, Krok, SD, Rizeni)

%SHOR2 Shorův r-algoritmus (heuristická modifikace)
%
%   [xmin, iter, kod] = Shor2(Funkce, Krok, SD, CasLimit, Rizeni)
%
%Funkce   Struktura {
%         f; cenová funkce
%         x0; počáteční aproximace
%         zeta; Poloměr okolí pro výpočet numerického subgradientu
%         fmin; Hodnota minima (volitelný parametr)
%         }
%
%Krok     Struktura {
%         h; Základní délka kroku
%         gamma; Koeficient 0 < gamma < 1 pro zmenšení délky kroku
%         my; Koeficient my >= 1 pro zvětšení délky kroku
%         L; Počet zvětšení kroku do zvětšení jeho základní délky
%         e; Vektor přesnosti
%              e = (e1, e2, e3, e4, e5)
%              --- Zadat požadované hodnoty ---
%              e1 = velikost subgradientu
%              e2 = velikost transformovaného subgradientu
%              e3 = délka kroku
%              e4 = vzdálenost dvou po sobě jdoucích aproximací
%                   || x(k+1) - x(k) ||
%              e5 = Velikost rozdílu hodnot minima aproximace minima
%                   | fmin - f(x(k)) |
%              Poznámka: Pokud některé kritérium nepoužijeme,
%              pak jeho hodnota musí být nastavena na -1
%         }
%
%SD       {
%         alfa; Koeficient dilatace prostoru
%         p; Počet iterací - interval optimalizace matice B
%         delta; Nejmenší přípustná hodnota největšího prvku |B|
%         ny; Koeficient ny > 1 pro násobení matice B
%         }
%
%Rizeni   Struktura {
```

```
%          tMax6_25; Max délka běhu algoritmu 6.25 [sekundy];
%          tMax7_1; Délka běhu algoritmu 7.1 [sekundy];
%          Opakovani; Počet pokusů o minimalizaci
%          DX; Maximální velikost norm(dx)
%          DF; Maximální velikost abs(df)
%          StIter; Od které iterace se má provádět kontrola
%          Stred; Souřadnice, od které měříme DX
%          }
%
%xmin    Argument minima cenové funkce
%iter    Počet provedených iterací
%kod     Kód zastavovací podmínky podle vektoru chyb e, navíc
%        100 = Řádný běh a dokončení bez restartu
%        101 = Řádný běh a dokončení s restartem
%        200 = Běh ukončen časovým limitem, ale výsledek splňuje kritéria
%        400 = Chybný výsledek

%Nastavení společné pro všechna opakování
casMax = Rizeni{2};
MaxPokusy = Rizeni{3};
kappa = 1/SD{1} - 1;
Presnost = Krok{5};
n = length(Funkce{2});
I = eye(n);
iter = 0;
StIter = Rizeni{6};
%Začátek série minimalizací
for m = 1 : MaxPokusy

    %Inicializace konkrétní minimalizace
    B = I;
    if m == 1 %první pokus
        xmin = Funkce{2};
        kod = 100;
    else %opakovaný pokus při chybném výsledku
        xmin = (Funkce{2} + 0.1) * 4 .* (rand(n,1) - 0.5);
        kod = 101;
    end

    %Výpočet 0. iterace
    gv = NumGrad(Funkce{1}, xmin, Funkce{3});
    if norm(gv) < Presnost(2)
        break; %Nulový subgradient -> máme výsledek
    end

    F = {Funkce{1}, Funkce{2}, Krok{1}, Krok{2}, Krok{3}, ...
        Krok{4},Rizeni{1}};
    AproxVariabilni2('set', F);

    eta = - gv / norm(gv);

    [xmin, ~] = AproxVariabilni2('provoz', {eta}); %První aproximace
    iter = iter + 1;


    %Výpočet dalších iterací
```

```
tic; %Začátek měření času algoritmu 7.1

while 1

    xmin2 = xmin;

    g = NumGrad(Funkce{1}, xmin, Funkce{3}); %Gradient
    if Presnost(1) > norm(g)
        break;
    end

    gh = B' * g; %Transformovaný subgradient
    if norm(gh) < Presnost(2)
        break;
    end

    r = gh - gv; %Rozdíl směrů
    rn = norm(r);
    if rn < Presnost(2)
        break;
    end
    ksi = r/rn;

    B = B * ( I + (kappa * (ksi * ksi') ) ); %Nová matice B

    if mod(iter,SD{2}) == 0 && iter ~= 0
        B = OptiB(B, SD{3}, SD{4}); %Optimalizace matice B
    end

    gv = B' * g;
    if norm(gv) < Presnost(2)
        break;
    end

    eta = -B * gv; %Směr minimalizace

    [xmin, krok] = AproxVariabilni2('provoz', {eta}); %Nová aprox.
    iter = iter + 1;

    %Ověření získaného řešení
    if iter > StIter

        Overeni = OverVysledek(xmin, Funkce, Rizeni);

        if Overeni == 0
            kod = 400; %Aproximace mimo povolený rozsah
            break;
        end

        if toc > casMax
            kod = 200; %Překročen časový limit
            break;
        end
```

```
        end


        if Presnost(3) > krok %Test délky kroku
            break;
        end

        dx = norm(xmin2 - xmin); %Test vzdálenosti dvou aproximací
        if Presnost(4) > dx
            break;
        end

        if Presnost(5) ~= -1
            df = abs(Funkce{1}(xmin) - Funkce{4});
            if Presnost(5) > df %Test rozdílu funkčních hodnot
                break;
            end
        end

    %pro úlohu Rosen 4D
%     if iter > 1000
%         kod = 400;
%         break;
%     end

    end %konec while


    %Ověření, zdali se má provést další pokus o minimalizaci

    if kod == 400 || kod == 200
        continue;
    end

    Overeni = OverVysledek(xmin, Funkce, Rizeni);
    if Overeni == 0
        kod = 400;
        continue;
    end

    break; %Máme výsledek, není potřeba opakování minimalizace

end %Konec cyklu for

if kod == 400 %Přiřazení nekonečna chybnému výsledku
    xmin = xmin + Inf;
    %xmin = xmin + NaN; %Pro úlohu Rosen 4D
end

end
```

Zdrojový kód: Testování stability výpočtu pro různé koeficienty dilatace prostoru

D. Zdrojový kód řešených úloh

Zdrojový kód řešených úloh slouží především jako ukázka volání implementovaných metod. Naším cílem určitě není zde uvádět vyčerpávající kód pro všechny varianty řešení. Z tohoto důvodu je ukázáno řešení pro jednu variantu s komentářem, jak postupovat dále. Identifikátory proměnných se zpravidla shodují s názvy uvedenými v textu i algoritmech. Značky řeckých písmen převádíme na jejich názvy.

1. Příklad 6.32

Soubor Priklady/Priklad_6_32.m

```
%Příklad 6.32

%Cenová funkce
f = @(x) 3*x(1).^2 + 2*x(2).^2 - x(1).*x(2) + 30*(abs(x(1)+x(2)) ...
    + abs(x(1)-x(2)));
xpr = [0;0];
fxpr = 0;

%Počáteční aproximace
x01 = [0;0];
x02 = [5;5];
x03 = [1e6; 1e6];

%Parametr c
c1 = 7.5;
c2 = 15;
c3 = 3e6;

%Vektory přesnosti
e1 = [1e-4, -1, 1e-4, -1, -1];
e2 = [1e0, -1, 1e0, -1, -1];

%Další parametry
zeta = 1e-2;
h = 100;
gamma = 0.5;
my = 2;
L = 5;

%Nastavení
Vysledky = {}; % spustíme pouze jednou, poté zakomentujeme!
x0 = x01;
c = c1;
e = e1;
Funkce = {f, x0, zeta};
Metoda = 'variabilni'; % nebo 'apriori'
%Data = {c, e}; % pro 'apriori'
```

```
Data = {h, gamma, my, L, e}; % pro 'variabilni'

%Výpočet
%Postupně zkoušíme různá nastavení
i = 1;
tic
[xmin, iter, kod] = NejSpad(Funkce, Metoda, Data);
Vysledky{i}{5} = toc;
Vysledky{i}{1} = norm(xmin - xpr);
Vysledky{i}{2} = abs(f(xmin) - fxpr);
Vysledky{i}{3} = kod;
Vysledky{i}{4} = iter;
disp(Vysledky{i});
```

Zdrojový kód: Příklad 6.32

2. Příklad 6.56

```
Soubor Priklady/Priklad_6_56.m

%Příklad 6.56

%Cenová funkce
f = @(x) 3*x(1).^2 + 2*x(2).^2 - x(1).*x(2) + 30*(abs(x(1)+x(2)) ...
    + abs(x(1)-x(2)));
zeta = 1e-2;
xpr = [0;0];
fxpr = 0;

%Počáteční aproximace
x01 = [0;0];
x02 = [5;5];
x03 = [1e6; 1e6];

%Vektory přesnosti
e1 = [1e-4, 1e-8, 1e-6, 1e-5, -1];
e2 = [1e-8, 1e-16, 1e-9, 1e-9, -1];
e3 = [1e-12, 1e-24, 1e-14, 1e-14, -1];

%Parametry délky kroku
h1 = 0.1;
h2 = 100;
gamma = 0.1;
my = 1.25;
L = 5;

%Parametry dilatace prostoru
alfa = 2.2;
p = 10;
delta = 1;
ny = 10;
```

```
%Nastavení
Vysledky = {}; % spustíme pouze jednou, poté zakomentujeme!
x0 = x03;
e = e3;
h = h2;
Funkce = {f, x0, zeta};
Krok = {h, gamma, my, L, e};
SD = {alfa, p, delta, ny};

%Výpočet - Postupně zkoušíme různá nastavení
i = 7;
tic
[xmin, iter, kod] = SDG(Funkce, Krok, SD);
Vysledky{i}{5} = toc;
Vysledky{i}{1} = norm(xmin - xpr);
Vysledky{i}{2} = abs(f(xmin) - fxpr);
Vysledky{i}{3} = kod;
Vysledky{i}{4} = iter;
disp(Vysledky{i});
```

Zdrojový kód: Příklad 6.56

3. Příklad 6.57

```
Soubor Priklady/Priklad_6_57.m

%Příklad 6.57

%Cenová funkce
f = @(x) 3*x(1).^2 + 2*x(2).^2 - x(1).*x(2) + 30*(abs(x(1)+x(2)) ...
    + abs(x(1)-x(2)));
zeta = 1e-2;
xpr = [0;0];
fxpr = 0;

%Počáteční aproximace
x0 = [5;5];

%Vektory přesnosti
e1 = [1e-20, -1, -1, -1, 1e-4];
e2 = [1e-20, 1e-20, -1, -1, 1e-4];

%Parametry délky kroku
h = 0.1;
gamma = 0.1;
my = 1.25;
L = 5;
c = 15;

%Parametry dilatace prostoru
```

```
alfa = 2.2;
p = 10;
delta = 1;
ny = 10;

%Nastavení
%Vysledky = {}; % spustíme pouze jednou, poté zakomentujeme!
e = e1;
Funkce = {f, x0, zeta, fxpr};
Krok = {h, gamma, my, L, e};
SD = {alfa, p, delta, ny};
Metoda = 'variabilni'; % nebo 'apriori'
%Data = {c, e}; % pro 'apriori'
Data = {h, gamma, my, L, e}; % pro 'variabilni'

%Výpočet - Postupně zkoušíme různá nastavení
i = 2;
tic
%[xmin, iter, kod] = NejSpad(Funkce, Metoda, Data);
[xmin, iter, kod] = SDG(Funkce, Krok, SD);
Vysledky{i}{4} = toc;
Vysledky{i}{1} = norm(xmin - xpr);
Vysledky{i}{2} = kod;
Vysledky{i}{3} = iter;
disp(Vysledky{i});
```

Zdrojový kód: Příklad 6.57

4. Příklad 7.3

```
Soubor Priklady/Priklad_7_3.m

%Příklad 7.3

%Cenová funkce
f = @(x) 3*x(1).^2 + 2*x(2).^2 - x(1).*x(2) + 30*(abs(x(1)+x(2)) ...
    + abs(x(1)-x(2)));
zeta = 1e-4;
xpr = [0;0];

%Počáteční aproximace
x0i = {[5;5], [-10;10], [10;0], [5;-50], [20;-17]};


%Vektory přesnosti
e1 = [1e-3, 1e-9, -1, 1e-4, -1];
e2 = [1e-6, 1e-18, -1, 1e-7, -1];
e3 = [1e-9, 1e-27, -1, 1e-10, -1];
e4 = [1e-12, 1e-36, -1, 1e-14, -1];
e5 = [1e-15, 1e-45, -1, 1e-17, -1];
ej = {e1, e2, e3, e4, e5};
```

```
%Parametry délky kroku
h = 0.1;
gamma = 0.1;
my = 1.25;
L = 5;

%Parametry dilatace prostoru
alfa = 3;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%Nastavení
Vysledky = cell(3,5,5,4); % spustíme pouze jednou, poté zakomentujeme!
m = 3;

for i = 1:5
    x0 = x0i{i};
    Funkce = {f, x0, zeta};
    for j = 1:5
        e = ej{j};
        Krok = {h, gamma, my, L, e};
        %Výpočet - Postupně zkoušíme různá nastavení
        tic
        [xmin, iter, kod] = Shor(Funkce, Krok, SD);
        Vysledky{m,i,j,1} = toc;
        Vysledky{m,i,j,2} = iter;
        Vysledky{m,i,j,3} = kod;
        Vysledky{m,i,j,4} = norm(xmin-xpr);
        disp(xmin);
    end
end
```

Zdrojový kód: Příklad 7.3

5. Rosenbrock – 1. část

```
Soubor Experimenty/Rosenbrock.m

%Rosenbrock

%Cenová funkce
f = @(x) 100*(x(2)-x(1).^2).^2 + (1-x(1)).^2;
xpr = [1;1];
fxpr = 0;
zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Počáteční aproximace
```

```
x0 = [0;0];
MAXiter = 100;

Funkce = {f, x0, zeta, xpr};

%Parametry délky kroku
h = 0.1;
gamma = 0.1;
my = 1.5;
L = 1e5;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
alfa = 2.5;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%-------------------------------------------------------------

%Závislost chyby aproximace na počtu iterací, maximální přesnost
Data_Rosenbrock = cell(1,3);
[Data_Rosenbrock{1}, Data_Rosenbrock{2}, Data_Rosenbrock{3}] = ...
    CasovaNarocnost(Funkce, Krok, SD, MAXiter);

%-------------------------------------------------------------

%Stabilita výpočtu - počáteční aproximace

tic
PoleDX = zeros(31,51);
PoleIT = zeros(31,51);
for j = 1 : 31
    parfor k = 1 : 51
        [xmin, iter, ~] = Shor({f, [1+0.1*(j-16);1+0.1*(k-16)], ...
            zeta}, Krok, SD);
        PoleDX(j,k) = norm(xmin-xpr);
        PoleIT(j,k) = iter;
    end
    disp(j);
end
Stab_PA_Rosenbrock = cell(1,2);
Stab_PA_Rosenbrock{1} = PoleDX;
Stab_PA_Rosenbrock{2} = PoleIT;

cas = toc;
disp(cas);

Chyby = 0;
for i = 1:31
    for j = 1:51
        if PoleDX(i,j)> 6
            PoleDX(i,j) = 0;
            Chyby = Chyby + 1;
```

```
        end
    end
end

DeltaXmax = max(PoleDX,[],'all');
disp(DeltaXmax);

MinIT = min(PoleIT,[],'all');
MaxIT = max(PoleIT,[],'all');

CelkemIteraci = sum(PoleIT,'all');

%-----------------------------------------------------------------

%Stabilita výpočtu - koeficient dilatace prostoru
PoleSDC_DX = zeros(1,201);
PoleSDC_IT = zeros(1,201);
parfor k = 1 : 201
    [xmin, iter, ~] = Shor(Funkce, Krok, {1.5 + 0.01*(k-1), p, ...
        delta, ny});
    PoleSDC_DX(k) = norm(xmin-xpr);
    PoleSDC_IT(k) = iter;
end
Stab_SDC_Rosenbrock = cell(1,2);
Stab_SDC_Rosenbrock{1} = PoleSDC_DX;
Stab_SDC_Rosenbrock{2} = PoleSDC_IT;
```

Zdrojový kód: Rosenbrock – 1. část

6. Rosenbrock – 2. část

```
Soubor Experimenty/Rosenbrock2.m

%Rosenbrock2

%Cenová funkce
f = @(x) 100*(x(2)-x(1).^2).^2 + (1-x(1)).^2;
xpr = [1;1];
fxpr = 0;
zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Počáteční aproximace
x0 = [0;0];
MAXiter = 100;

Funkce = {f, x0, zeta};

%Parametry délky kroku
h = 0.1;
gamma = 0.1;
```

```
my = 1.5;
L = 1e4;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
alfa = 2.5;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%Parametry struktury Řízení
tMax6_25 = 0.10;
tMax7_1 = 2.0;
Opakovani = 10;
DX = 5*sqrt(34);
DF = -1;
StIter = 10;
Stred = [1;2];
Rizeni = {tMax6_25, tMax7_1, Opakovani, DX, DF, StIter, Stred};


%-------------------------------------------------------------

%Stabilita výpočtu - počáteční aproximace
tic
PoleDX = zeros(31,51);
PoleIT = zeros(31,51);
for j = 1 : 31
    parfor k = 1 : 51
        [xmin, iter, ~] = Shor2({f, [1+0.1*(j-16);1+0.1*(k-16)], ...
            zeta}, Krok, SD, Rizeni);
        PoleDX(j,k) = norm(xmin-xpr);
        PoleIT(j,k) = iter;
    end
    disp(j);
end
Stab_PA_Rosenbrock2 = cell(1,2);
Stab_PA_Rosenbrock2{1} = PoleDX;
Stab_PA_Rosenbrock2{2} = PoleIT;

cas = toc;
disp(cas);

OznaceneChyby = 0;
NeoznaceneChyby = 0;
for i = 1:31
    for j = 1:51
        if PoleDX(i,j)== inf
            OznaceneChyby = OznaceneChyby + 1;
            PoleDX(i,j) = 0;
        elseif PoleDX(i,j) > 6
                PoleDX(i,j) = 0;
                NeoznaceneChyby = NeoznaceneChyby + 1;
        end
    end
```

```
end

PrumernaChyba = sum(PoleDX,'all') / 1581;
CelkemIteraci = sum(PoleIT,'all');
PrumerneIteraci = CelkemIteraci / 1581;

%-------------------------------------------------------------

%Stabilita výpočtu - koeficient dilatace prostoru
PoleSDC_DX = zeros(1,201);
PoleSDC_IT = zeros(1,201);
parfor k = 1 : 201
    [xmin, iter, ~] = Shor2(Funkce, Krok, {1.5 + 0.01*(k-1), p, ...
        delta, ny},Rizeni);
    PoleSDC_DX(k) = norm(xmin-xpr);
    PoleSDC_IT(k) = iter;
end
Stab_SDC_Rosenbrock2 = cell(1,2);
Stab_SDC_Rosenbrock2{1} = PoleSDC_DX;
Stab_SDC_Rosenbrock2{2} = PoleSDC_IT;
```

Zdrojový kód: Rosenbrock – 2. část

7. Půlměsíc

```
Soubor Experimenty/Pulmesic.m

%Půlměsíc

%Cenová funkce
f = @(x) max(x(1).^2 + (x(2)-1).^2 + x(2) -1, -x(1).^2 - ...
    (x(2)-1).^2 + x(2) + 1);
xpr = [0;0];
fxpr = 0;
zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Počáteční aproximace
x0 = [-2;-2];
MAXiter = 1000;

Funkce = {f, x0, zeta, xpr};

%Parametry délky kroku
h = 0.1;
gamma = 0.1;
my = 1.5;
L = 100;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
```

```
alfa = 2.5;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%Parametry struktury Řízení
tMax6_25 = 0.100;
tMax7_1 = 2.000;
Opakovani = 10;
DX = 30*sqrt(2);
DF = -1;
StIter = 10;
Stred = [0;0];
Rizeni = {tMax6_25, tMax7_1, Opakovani, DX, DF, StIter, Stred};

%-------------------------------------------------------------

%Závislost chyby aproximace na počtu iterací, maximální přesnost
Data_Pulmesic = cell(1,3);
[Data_Pulmesic{1}, Data_Pulmesic{2}, Data_Pulmesic{3}] = ...
    CasovaNarocnost(Funkce, Krok, SD, MAXiter);

%-------------------------------------------------------------

%Stabilita výpočtu - počáteční aproximace
tic
PoleDX = zeros(61,61);
PoleDF = zeros(61,61);
PoleIT = zeros(61,61);
for j = 1 : 61
    parfor k = 1 : 61
        [xmin, iter, ~] = Shor2({f, [0+0.1*(j-31);0+0.1*(k-31)], ...
            zeta}, Krok, SD, Rizeni);
        PoleDX(j,k) = norm(xmin - xpr);
        PoleDF(j,k) = abs(f(xmin)-f(xpr));
        PoleIT(j,k) = iter;
    end
    disp(j);
end
Stab_PA_e7_Pulmesic = cell(1,2);
Stab_PA_e7_Pulmesic{1} = PoleDX;
Stab_PA_e7_Pulmesic{2} = PoleIT;

cas = toc;

PoleDF2 = PoleDF;
PoleDX2 = PoleDX;

OznaceneChyby = 0;
NeoznaceneChyby = 0;
for i = 1:61
    for j = 1:61
        if PoleDX(i,j)== inf
            OznaceneChyby = OznaceneChyby + 1;
            PoleDX(i,j) = 0;
```

```
            PoleDF(i,j) = 0;
        elseif PoleDX(i,j) > 2 * sqrt(3)
                PoleDX(i,j) = 0;
                PoleDF(i,j) = 0;
                NeoznaceneChyby = NeoznaceneChyby + 1;
        end
    end
end

DeltaXmax = max(PoleDX,[],'all');
MinIT = min(PoleIT,[],'all');
MaxIT = max(PoleIT,[],'all');
CelkemIteraci = sum(PoleIT,'all');

disp(DeltaXmax);
disp([OznaceneChyby NeoznaceneChyby]);
disp(100*OznaceneChyby/3721);
disp(1000*cas/3721);
disp(sum(PoleDF,'all')/(3721-OznaceneChyby));
disp(median(PoleDF2,'all'));
disp(median(PoleDX2,'all'));

%-------------------------------------------------------------

%Stabilita výpočtu - koeficient dilatace prostoru
PoleSDC_DX = zeros(1,201);
PoleSDC_IT = zeros(1,201);
parfor k = 1 : 201
    [xmin, iter, ~] = Shor2(Funkce, Krok, {1.5 + 0.01*(k-1), p, ...
        delta, ny}, Rizeni);
    PoleSDC_DX(k) = norm(xmin-xpr);
    PoleSDC_IT(k) = iter;
end
Stab_SDC_Pulmesic = cell(1,2);
Stab_SDC_Pulmesic{1} = PoleSDC_DX;
Stab_SDC_Pulmesic{2} = PoleSDC_IT;
```

Zdrojový kód: Půlměsíc

8. Charalambous-Bandler – 1. část

```
Soubor Experimenty/CharalambousBandler.m

%Charalambous-Bandler

%Cenová funkce
f1 = @(x) x(1).^4 + x(2).^2;
f2 = @(x) (2-x(1)).^2 + (2-x(2)).^2;
f3 = @(x) 2 * exp(-x(1)+x(2));
f = @(x) max(f1(x), max(f2(x), f3(x)));
```

```
xpr = [1;1];
fxpr = 2;
zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Počáteční aproximace
x0 = [0;2];
MAXiter = 1000;

Funkce = {f, x0, zeta, xpr};

%Parametry délky kroku
h = 0.10;
gamma = 0.10;
my = 1.5;
L = 2500;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
alfa = 2.5;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%---------------------------------------------------------------

%Závislost chyby aproximace na počtu iterací, maximální přesnost
Data_CharalambousBandler = cell(1,3);
[Data_CharalambousBandler{1}, Data_CharalambousBandler{2}, ...
    Data_CharalambousBandler{3}] = CasovaNarocnost(Funkce, Krok,...
    SD, MAXiter);
%---------------------------------------------------------------
```

Zdrojový kód: Charalambous-Bandler – 1. část

9. Charalambous-Bandler – 2. část

```
Soubor Experimenty/CharalambousBandler2.m

%Charalambous-Bandler2

%Cenová funkce
f1 = @(x) x(1).^4 + x(2).^2;
f2 = @(x) (2-x(1)).^2 + (2-x(2)).^2;
f3 = @(x) 2 * exp(-x(1)+x(2));
f = @(x) max(f1(x), max(f2(x), f3(x)));

xpr = [1;1];
fxpr = 2;
zeta = 1e-4;
```

```
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Počáteční aproximace
x0 = [0;2];
MAXiter = 1000;

Funkce = {f, x0, zeta};

%Parametry délky kroku
h = 0.1;
gamma = 0.10;
my = 1.25;
L = 10; %Měníme na 10, 20, 50 a 100
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
alfa = 1.0;
p = 10;
delta = 1;
ny = 20;
SD = {alfa, p, delta, ny};

%Parametry struktury Řízení
tMax6_25 = 0.100;
tMax7_1 = 2.000;
Opakovani = 50;
DX = 25;
DF = -1;
StIter = 10;
Stred = [0.5;1];
Rizeni = {tMax6_25, tMax7_1, Opakovani, DX, DF, StIter, Stred};

%---------------------------------------------------------------

%Závislost charakteristik výpočtu na koeficientu alfa
AnalyzaCharalBandler = cell(4,31); %Spustíme pouze jednou
Mereni = 1; %Postupně zvyšujeme 1,2,3,4

for i = 1 : 31

    disp(i);
    vysledek = zeros(8,1);
    alfa = 1 + (i-1)*0.10;
    vysledek(1) = alfa;
    SD{1} = alfa;
    tic

    PoleDX = zeros(16,21);
    PoleDF = zeros(16,21);
    PoleIT = zeros(16,21);
    for j = 1 : 16
        parfor k = 1 : 21
            [xmin, iter, ~] = Shor2({f, [0.5+0.2*(j-8);1+0.2*(k-10)],...
                zeta}, Krok, SD, Rizeni);
            PoleDX(j,k) = norm(xmin-xpr);
```

```
            PoleDF(j,k) = abs(f(xmin)-f(xpr));
            PoleIT(j,k) = iter;
        end
    end

    vysledek(2) = toc / 1271;
    vysledek(3) = sum(PoleIT,'all') / 1271;

    OznaceneChyby = 0;
    NeoznaceneChyby = 0;
    for j = 1:16
        for k = 1:21
            if PoleDX(j,k)== inf
                OznaceneChyby = OznaceneChyby + 1;
                PoleDX(j,k) = 0;
                PoleDF(j,k) = 0;
            elseif PoleDX(j,k) > 2.5
                PoleDX(j,k) = 0;
                PoleDF(j,k) = 0;
                NeoznaceneChyby = NeoznaceneChyby + 1;
            end
        end
    end

    vysledek(4) = OznaceneChyby;
    vysledek(5) = NeoznaceneChyby;
    Spravne = 1271 - (OznaceneChyby + NeoznaceneChyby);
    vysledek(6) = Spravne / 1271;
    vysledek(7) = sum(PoleDX,'all') / Spravne;
    vysledek(8) = sum(PoleDF,'all') / Spravne;
    AnalyzaCharalBandler{Mereni,i} = vysledek;

end
%-------------------------------------------------------------
```

Zdrojový kód: Charalambous-Bandler – 2. část

10. Maxl – Dimenze 20

```
Soubor Experimenty/Maxl.m

%Maxl - Dimenze 20
dim = 20;

%Cenová funkce
f = @(x) max(abs(x));

xpr = zeros(dim,1);
fxpr = 0;
zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];
```

```
%Počáteční aproximace
cast1 = round(dim / 2);
x0 = [1:cast1, -(cast1+1):-1:-dim]';
MAXiter = 1000;

Funkce = {f, x0, zeta, xpr};

%Parametry délky kroku
h = 0.1;
gamma = 0.10;
my = 1.50;
L = Inf;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
alfa = 1.50;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%-------------------------------------------------------------

tic
%Závislost chyby aproximace na počtu iterací, maximální přesnost
Data_Maxl20 = cell(1,3);
[Data_Maxl20{1}, Data_Maxl20{2}, Data_Maxl20{3}] = ...
    CasovaNarocnost(Funkce, Krok, SD, MAXiter);
disp(toc);
%-------------------------------------------------------------
```

Zdrojový kód: Maxl – dimenze 20

11. Maxl – Závislost výpočtu na dimenzi úlohy

```
Soubor Experimenty/MaxlDim.m

%Cenová funkce
f = @(x) max(abs(x));

zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Parametry délky kroku
h = 0.1;
gamma = 0.10;
my = 1.50;
L = 10;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
```

```
alfa = 1.00;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%Parametry struktury Řízení
tMax6_25 = 0.100;
tMax7_1 = 2.000;
Opakovani = 1;
DX = -1;
DF = -1;
StIter = 1000;
Rizeni = {tMax6_25, tMax7_1, Opakovani, DX, DF, StIter};

%-------------------------------------------------------------

%Závislost chyby aproximace na počtu iterací, maximální přesnost

MaxlVar = zeros(5,350);

parfor i = 1 : 350
    cas = tic;
    pozx = round(i / 2);
    x0 = [1:pozx, -(pozx+1):-1:-i]';
    [xmin, iter, ~] = Shor2({f, x0, zeta}, Krok, SD,Rizeni);
    MaxlVar(:,i) = [i; iter; norm(xmin); f(xmin); toc(cas)];
end
%-------------------------------------------------------------
```

Zdrojový kód: Maxl – Závislost výpočtu na dimenzi úlohy

12. Neostré globální minimum

```
Soubor Experimenty/NeostreGMin.m

%Cenová funkce
f = @(x) abs(max(x) - sum(x));

fxpr = 0;
zeta = 1e-4;
e = [1e-6, 1e-18, -1, 1e-7, -1];

%Parametry délky kroku
h = 0.1;
gamma = 0.10;
my = 1.50;
L = 100;
Krok = {h, gamma, my, L, e};

%Parametry dilatace prostoru
```

```
alfa = 2;
p = 10;
delta = 1;
ny = 10;
SD = {alfa, p, delta, ny};

%Parametry struktury Řízení
tMax6_25 = 0.25;
tMax7_1 = 5.0;
Opakovani = 1;
DX = -1;
DF = -1;
StIter = 10;
Rizeni = {tMax6_25, tMax7_1, Opakovani, DX, DF, StIter};

%Počáteční aproximace
X0 = zeros(50,8);
X0(1,2) = 1;
X0(50,3) = 3;
X0(2:2:50,4) = 1;
X0(1:2:49,4) = -1;
X0(3,4) = 0;
X0(2,5) = -2;
X0(:,6) = 1;
X0(:,7) = 1:50;
X0(1:25,8) = -1 : -1 : -25;
X0(26:50,8) = 26 : 1 : 50;


%---------------------------------------------------------------

%Závislost výpočtu na počáteční aproximaci

Data_NeostreGMin = cell(4,8);

for i = 1:8
    [Data_NeostreGMin{1,i}, Data_NeostreGMin{2,i}, ~] = ...
        Shor2({f, X0(:,i), zeta}, Krok, SD, Rizeni);
    Data_NeostreGMin{3,i} = f(Data_NeostreGMin{1,i});
    Data_NeostreGMin{4,i} = norm(Data_NeostreGMin{1,i});
    disp(i);
end
%---------------------------------------------------------------
```

Zdrojový kód: Maxl – Závislost výpočtu na dimenzi úlohy